(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

UNITED STATES NAVY · NAVAL POSTGRADUATE SCHOOL ·

THESIS

REFAB:
A PROTOTYPE GRAPHICAL FRONT_END
FOR THE RESA NAVAL WARGAME

by

Thomas G. Avey

June 1990

Thesis Advisors: James N. Eagle
John M. Yurchak

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School

6b. OFFICE SYMBOL (if applicable): 52

7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School

6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000

7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000

8a. NAME OF FUNDING/SPONSORING ORGANIZATION

8b. OFFICE SYMBOL (if applicable)

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS (City, State, and ZIP Code)

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification): REFAB: A Prototype Graphical Frontend for the RESA Naval Wargame

12. PERSONAL AUTHOR(S): Avey, Thomas G.

13a. TYPE OF REPORT: Master's Thesis

13b. TIME COVERED: FROM ______ TO ______

14. DATE OF REPORT (Year, Month, Day): June 1990

15. PAGE COUNT: 77

16. SUPPLEMENTARY NOTATION: The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): RESA, Graphic User Interfaces, Computer Wargame Simulations, BATMAN and ROBIN, Man-Machine Interfaces

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Computer wargame simulations have typically provided military officers with an effective method of testing their knowledge of tactics and strategy. However, large simulations typically use a command language dialogue interfacing the user with the wargame. This type of interface requires a great deal of typing skill and memorization of the language syntax and allowing little time for decision making and battle analysis. RESA is a typical theater-level naval wargame which utilizes this typeof interface. Presented in this research is the RESA Enhanced FORCE and BUILD (REFAB), a first phase prototype development of a graphic user interface utilizing a bit-mapped display and windowing environment. REFAB was developed from an existing system, BATMAN and ROBIN. The interface concentrates on combinations of form filling, graphic, direct manipulation, and iconic dialogues, and stresses information presentation. This interface could be utilized for RESA to ease the system operating requirements placed on the RESA users, allowing the user to make timely decisions, gather information more quickly, and provide a more rewarding wargaming session.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL: Eagle, James N. and Yurchak, John M.

22b. TELEPHONE (Include Area Code): (408) 646-2654

22c. OFFICE SYMBOL: OR/ER

Approved for public release; distribution is unlimited.

REFAB:
A PROTOTYPE GRAPHICAL FRONTEND
FOR THE RESA NAVAL WARGAME

by

Thomas Gregg Avey
Captain, United States Marine Corps
B.S., University of Utah

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1990

Author: ______________________
Thomas Gregg Avey

Approved By: ______________________
James N. Eagle, Thesis Advisor

____
John M. Yurchak, Thesis Advisor

____
Robert B. McGhee, Chairman,
Department of Computer Science

ABSTRACT

Computer wargame simulations have typically provided military officers with an effective method of testing their knowledge of tactics and strategy. However, large simulations typically use a command language dialogue interfacing the user with the wargame. This type of interface requires a great deal of typing skill and memorization of the language syntax and allows little time for decision making and battle analysis. RESA is a typical theater-level naval wargame which utilizes this type of interface. Presented in this research is the RESA Enhanced FORCE and BUILD (REFAB), a first phase prototype development of a graphic user interface utilizing a bit-mapped display and windowing environment. REFAB was developed from an existing system, BATMAN and ROBIN. The interface concentrates on combinations of form filling, graphic, direct manipulation, and iconic dialogues, and stresses information presentation. This interface could be utilized for RESA to ease the system operating requirements placed on the RESA users, allowing the user to make timely decisions, gather information quickly, and provide a more rewarding wargaming session.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank the following individuals for their support in this research and the development of REFAB:

- LCDR Yurchak for his encouragement, insight, assistance, and overlooking all of my interruptions,
- Capt Block for his support, funding of this research effort, and an occasional ear which helped to blow off steam,
- Jeff Franklin, Alan Jones, and Bill Thomas for their assistance in the WARLAB,
- Chuck Lombardo for his last minute assistance and tutoring in Framemaker, and
- Rosalie Johnson and Sue Whalen for their technical assistance in customizing my development environment.

But most importantly, I thank my wife, Cynthia and my daughter, Catherine, for their patience and support during the past twenty-two months at the Naval Postgraduate School. Without their support and understanding, this achievement would not have been possible.

I. INTRODUCTION

Computer wargaming simulations have typically provided military officers with an effective method of testing their knowledge of tactics and strategy. But they are also typically awkward to use, relying on an often cryptic syntax-directed command language. The goal of this thesis is to identify the requirements and produce a prototype of a more intuitive, simple, user-controlled interface for a computer wargame simulation.

A. BACKGROUND

Wargaming is a vital educational and research tool for the tactical training of today's officers. When employed as a computer simulation, it is an effective method for testing doctrine, strategy, tactics and practical decision making. Additionally, its economic benefits allow for the analysis of strategic plans and operations without the physical deployment of military forces. Careful analysis during simulation gives the user the foresight to "predict" the consequences of specific courses of actions. Increased interest and acceptance has produced a variety of computer simulations.

Unfortunately, this complex modeling environment has also produced user interfaces which are equally complex. User input is typically via a syntactic command line interface. Commands are input through the keyboard and are often very structured and cryptic. Any deviation from the command syntax whether through misspelling or an incorrect argument, requires the entire command to be re-entered. This gives the user the feeling that the system is in control of the user, instead of the user directing the system. Additionally, this type of interface requires the user to remember an extensive set of commands and syntax, requiring either in-depth training or providing operator personnel.

The Research, Evaluation and Systems Analysis (RESA) developed at the Naval Ocean Systems Center (NOSC), is a typical wargaming system used at the Naval Postgraduate

School. RESA is a multi-user interactive naval wargame used for wargaming, research, teaching and battle group tactical training. It is written in the RATional FORtran language, RATFOR, and runs under the VMS operating system on a VAX 11/780. RESA utilizes both VT100/102 type terminals for command input and status boards, and RAMTEK graphics displays for theater display. It is designed as a two-sided (Blue and Orange forces) game whereby users direct platforms or objects (aircraft, ships, subs, missiles) in a real-time battle scenario. It also provides an umpire control capability of neutral forces.

RESA is used in conjunction with several courses at the Naval Postgraduate School and the users are typically student officers. They have no previous knowledge of RESA and little, if any, experience with wargaming or computers. Additionally, time constraints and class schedules provides little time for training. The interface for RESA is very inefficient for this class of users, the novice.

B. SCOPE

RESA's inefficiency stems from it's out-dated user interface. The three main areas of "unfriendliness" are:

- Keyboard entry - All input is via the keyboard. With users who are typically unfamiliar with wargames and computers, this can be a hinderance. Although RESA provides some assistance by accepting abbreviated commands if unique enough to distinguish from other commands, the impetus is still on the user to correctly type the commands. In tense situations as may appear during a wargame, misspellings can cause a great deal of frustration.
- Knowledge of Command Syntax - While RESA commands are fairly descriptive of their actions, the user is still required to commit to memory, a complex sequence of commands. This requires the use of on-line help which RESA provides. But this draws the user away from his current input, and a player can become lost in the command structure. And again, in tense situations, the user is not going to want to stop to look something up in the on-line help.
- Variety of Information on a Variety of Terminals - As previously stated, RESA utilizes VT100/102 terminals for command input and computer response. Additionally, the VT100 terminals are used to provide the user with current data on

game platforms. And the RAMTEK graphic displays are used to provide a current overhead view of the current game situation. The user must swing his attention from one terminal to the next to "find" the information required.

As can be seen, RESA is not a wargame for first-time or sporadic use. It is not that RESA is of a poor design, but that technology has passed it by. RESA has been under continuing development for several users. With the increased use of bit-mapped displays and window management systems, many systems have been developed which provide the user with a more intuitive environment in which to operate. Very little training is needed to advance the user from the novice level. It is now time for RESA and similar wargames to catch up to technology and become more "user friendly". The following subjective tasks can be used to measure user friendliness:

- time to learn
- speed of performance
- rate of errors by users
- subjective satisfaction
- retention over time (Shneiderman, 1987, pp. 73-74)

Enhancing the user interface by optimizing the previously mentioned measures, could increase user productivity. By utilizing an interface which is more "user friendly" and intuitive, users could complete tasks more quickly and efficiently, with less aggravation and concentrate more on decision making tasks.

RESA is separated into five major processes:

- BUILD - establishes the detailed database of characteristics for all force, sensor, weapon, and C3 system objects.
- FORCE - establishes the exercise initial conditions for the wargame, responds to Force orders and operates on subsets of the characteristics database.
- Wargame Initialization - initializes the master database tables or blackboard of the warfare environment for game play.
- The Wargame - executes user orders, operates models, updates the blackboard, and outputs platform status tables.
- Postgame Analysis - collects and processes warfare environment data and provides tables of analysis

The focus of this thesis will be on the first two pregame processes, BUILD and FORCE. An implementation of a RESA game session involves the creation of a game file to identify the opposing forces, Blue forces consisting of one or more carrier battle groups, Orange forces (surface action groups, submarines, land based attack and fighter aircraft), and environmental factors (acoustic and electronic propagation, noise conditions, and weather). This input is accomplished through the RESA frontend, FORCE and BUILD. Pre-game setup is a time consuming process. Exercise scenarios typically take a week to generate from scratch. With this time constraint, players never have the opportunity to generate their own scenarios and must rely on the staff to provide this function. Even then, staff generated scenarios are typically previously generated scenarios, modified to accept new situations. They are very rarely designed from scratch. By enhancing the capabilities for defining exercise scenarios, players could become more involved in game setup and the staff could create additional scenarios from scratch in a more timely manner.

The objective of this thesis will be to apply design guidelines for man-machine interfaces to the development of a prototype which is more "user friendly" than the current RESA. The RESA Enhanced FORCE And BUILD (REFAB) interface will stress information presentation allowing the user to make timely decisions, gather information more quickly, and ease human-computer communications.

II. ANALYSIS OF USER INTERFACE REQUIREMENTS

A. BUILD

As stated previously, the BUILD function is utilized to create a database of static force, sensor, and weapons characteristics. These "object" characteristics describe the simulation platforms and systems, and may be real, proposed or notional. This flexibility allows for the testing and integrating of new, modified and proposed platforms within the current strategic and tactical doctrine, allowing users to visualize their effects, strengths and weaknesses. Force characteristics describe ship classes (surface, submarine, small boat), aircraft types, and shore bases. Sensor characteristics are used to model radars, ESM systems, jammers (communications/radar), active/passive sonars, sonobuoys, surveillance satellites, HFDF systems, SOSUS systems, communications equipment/buoys, and navigation systems. Weapons characteristics define cruise missiles, torpedoes, AAM, SAM, CIWS, bombs, guns, and mines. The BUILD database consists of a set of fifteen files. Each file name starts with the same five alphanumeric characters, the database name. This is followed by "C." and a three letter designation for the type of objects defined in that file. The object characteristics are identified in Appendix A. (NOSC, 1988)

The center of RESA is the set of models defining object behavior relative to other objects, the current situation and environmental factors. The models are essentially equations with the characteristics database providing the static or constant terms. The variables for the models are defined by the dynamic characteristics of the game situation.

For example, the RESA radar model for signal excess uses the following equation to calculate radar detections:

$$SE = PT + 2G + 2W + TCS - 4RD - B - NF - L - C$$

where:

SE = Signal excess or strength of the returning radar signal

PT = Peak transmitted power

G = Antenna gain

W = Wavelength

TCS= Radar cross section of the target

D = Ducting factor

R = Range to the target

B = IF bandwidth of the radar receiver

NF = Noise factor of the radar

C = Sea clutter factor

L = Radar system loss factor (NOSC, 1988, p. 2)

Sea clutter and ducting are environmental factors and as such, will depend on the current situation. The range to the target is also dynamic and is determined as needed. The variable TCS is obtained from the ship or aircraft database, depending on the target. The rest of the variables are constants drawn from the radar/ESM database (XXXXXC.SEN). It is important to note that any creation of objects within BUILD must specify all of the required static characteristics for that object.

BUILD functions as an interactive editor of the characteristics database. Once BUILD is invoked, the user is prompted for the name of the characteristics database, which is then loaded. The user can then prepare the database to fit the needs of the wargame scenario by utilizing three types of commands or "orders":

- BUILD program control orders

- Object (e.g. ships, weapons, radar) access orders
- Characteristics control orders.

The program control orders direct the top level of the BUILD program. These commands consist of the following functions:

- **BUILD** - This order loads the BUILD program, defines the database name and presents the top level menu of orders as depicted in Figure 1. At this point, any order on the menu may be entered. BUILD and FORCE require only enough of the order be entered which differentiates it from the other available orders.

```
----MESSAGES------------------------------------BUILD------



----INPUT--------------------------------------------------
NEXT INPUT (BYE,PRINT,AIR,COMMBUOY,COMMJAMMER,
 COMMPAIR,COMMSUITE,CRUISE MISSILE,JAMMER,NAVAID,RADAR/ESM,
 SURVSAT,SHIP,SHORE BASE,SONAR,SONOBUOY,or WEAPON)...
```

Figure 1 -BUILD Screen Layout

- **PRINT** - This order generates a file of the characteristics database for printing. If no parameter is defined, then the entire database is written to the file. If a name of one of the characteristics files is entered, then only that characteristics file is written. The name of the file generated is of the form XXXXXC.LIS.
- **WRITE** - This order copies contents of the characteristics database into a file which can then be edited by a standard text editor. If no parameter is defined, then the entire database is written to the file. If a name of one of the characteristics files is entered, then only that characteristics file is written. The name of the file generated is of the form XXXXXC.TXT.

- **BYE** - This order closes an open characteristics file and returns to the top level menu or if at the top level menu, terminates the BUILD program. Open files are not automatically saved when this order is executed.

The object access orders open their respective object characteristics file for editing. Only one characteristics file can be open at a time. These orders make available the characteristics control orders. Returning to the top level menu is accomplished through the program control order, **BYE**. The object access orders include the following: **AIR, COMMBUOY, COMMPAIR, COMMJAMMER, COMMSUITE, CRUISE MISSILE, JAMMER, NAVAID, RADAR/ESM, SURVSAT, SHIP, SHORE BASE, SONAR, SONOBUOY, WEAPON**. A listing of the required and optional characteristics is displayed on the terminal. A new instance of an object may then be entered into the database by providing all of the required characteristics. Any omissions are identified when the user tries to save the object.

The characteristics control orders provide the editing access to the objects and the their characteristics. These commands consist of the following functions:

- **FIND** - This order along with the parameter, object_name, will search the open characteristics file for an object matching the parameter. If a match is found, then a copy of that objects characteristics will be placed in the work area. The characteristics are also displayed on the display terminal. If another objects characteristics were in the work area when the **FIND** order was given, then those characteristics will be saved prior to the new characteristics being made available in the work area.
- **SAVE** - This function checks for the completeness of the characteristics for an object in the work area as defined by the database dictionary in Appendix A. If incomplete, then BUILD prompts for completion. Otherwise, the open characteristics file is updated and closed.
- **LIST** - This order lists the names of all objects in the currently open characteristics file.
- **MORE** - This order will save any characteristics in the work area if they have been changed. Additionally, it functions as a copy routine, where the name of the object currently in the work area is deleted, but the other characteristics remain. This allows the user to define multiple objects with similar characteristics. The object is named with the **NAME** order.

- **NAME** - This order functions with **MORE** as a copy routine, where a name may be assigned to the object currently in the work area.
- **CLASS** - This order provides the same function as **NAME**, but for the ship objects.
- **KILL** - This order clears the work area of all characteristics. The effect is to erase changes to the characteristics prior to saving. This order has no effect on the characteristics file.
- **DELETE** - This orders allows for the deletion of characteristics for the object currently in the work area. Single or multiple occurrences of characteristics may be identified for deletion as well as an entire set of characteristics for an object.
- **HELP** - This order provides explanatory comments for the named parameters. Help may be requested for any program order or the names of any characteristics of objects in the database.

The BUILD grammar for these orders is defined further in Appendix B.

B. FORCE

The FORCE program is utilized to initialize the scenario of the simulation. FORCE uses characteristics from the characteristics database modified using BUILD. Additionally, FORCE provides for the identification of the following operational situation characteristics: specific ships and shore bases, task hierarchy organization, definition of views, types and maintenance status of aircraft, communications networks, weather and acoustic environment, EMCON conditions, and restored commands. Characteristics from the BUILD database are mapped into a set of thirty-two scenario files as required by FORCE commands. These scenario files are later mapped to the RESA game blackboard through the wargame unitization process. The blackboard maintains the status and control of all objects in the warfare environment. The file name structure and data contained in these files is identified in Appendix A. (NOSC, 1989)

FORCE also functions as an interactive editor for the initial conditions of the game utilizing the objects available from the BUILD database. A user may bypass the interactive action of FORCE by feeding the set of FORCE orders to FORCE via a text file. This action will not be addressed in this thesis as it requires minimal user interaction.

FORCE recognizes two types of orders: program control orders and scenario control orders. FORCE program control orders provide the same function as do the program control orders for BUILD by controlling the top-level functions performed. The program control orders consist of the following commands:

- **FORCE** - This order loads the FORCE program. It prompts the user to define the BUILD database name, the name of the scenario and the nature of the scenario as follows:
    - **NEW** if a new scenario is to be created,
    - **UPDATE** if an existing scenario is to be used without referencing the BUILD database, i.e., changes made to the BUILD files since the scenario creation will not be reflected,
    - **RELINK** if an existing scenario is to be used and the BUILD database is to be re-linked to reflect any changes.

  Unlike the **BUILD** order, **FORCE** provides no menu of commands to the user as shown in Figure 2. At the *Command:* prompt, the user may type a question mark (?) at any time during command input and an appropriate help message will be displayed in the display area above.
- **BYE** - This order validates the FORCE orders made to create the scenario, saves the scenario, and terminates the FORCE program. If an error is detected while comparing the BUILD database and the FORCE scenario, the user is prompted for corrections.
- **STOP** - This order forces program termination whereby the current scenario is not saved.

The FORCE scenario control orders control the generation of objects, situations, and conditions that define the scenario. The syntax for the following control orders is defined in Appendix B:

- **ENTER** - This order generates an instance of an object for the scenario as determined by the parameters provided. Although these orders do not have to be entered in a particular order, the orders are presented in the order in which a user might create a scenario. For example, ships and shore bases are defined before aircraft can be placed on them.
- **DELETE** - This order deletes from the scenario the entities specified by the parameters.

• **PRINT** - This order writes portions of or the entire scenario to a disk file. The name of the file generated is of the form XXXXXS.LIS.

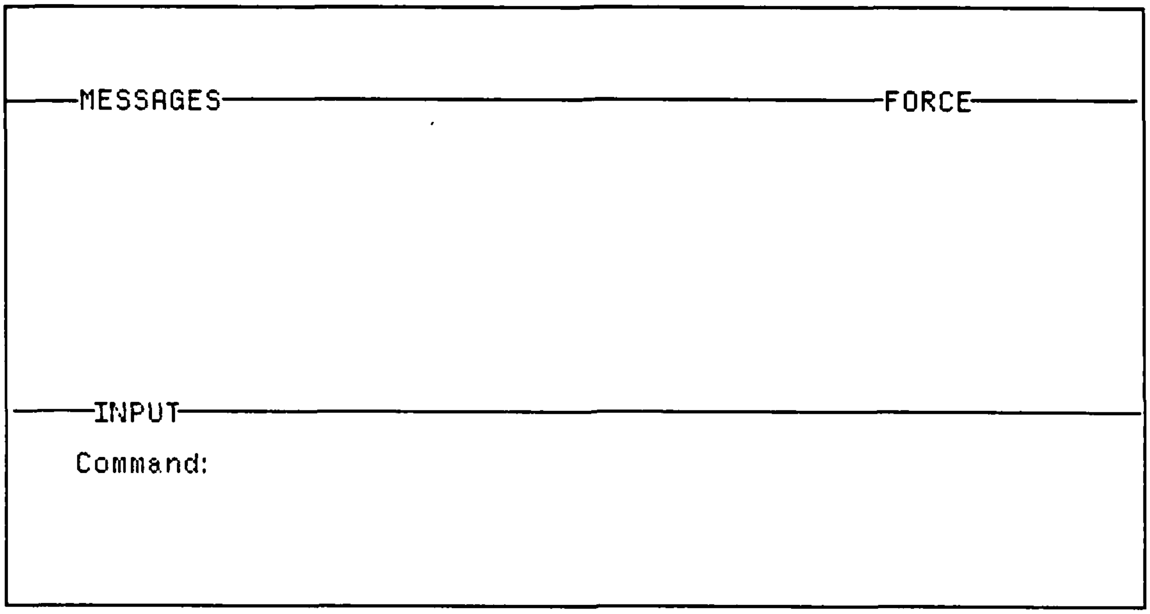

Figure 2 -FORCE Screen Layout

C. DESIGN ISSUES IN THE DEVELOPMENT OF REFAB

The goal of any user interface is to project a specific dialogue style on the user which is defined by a consistent and unified set of interaction techniques. There are various categories of interactive styles which may be used to insulate the user from the computer or software system. RESA currently uses a combination of command line dialogue and programming language dialogue. The command syntax is very rigid and the user is required to type the instructions to the computer in this structured pseudo-language. A scenario can be created from within a text editor using the force orders and input into FORCE non-interactively.

Window systems and environments make use of numerous overlapping or tiled windows which provide access to multiple sources of information. Each window can support its own individual interactive style or dialogue, thus allowing for the integration of multiple concurrent activities on the same display. But windows alone do not present the panacea of a visual environment. What is displayed in the windows and how it is presented to the user is the key.

Iconic images are a "natural" dialogue for users because the brain has a tremendous processing and storing capability for pictures. Icons are symbols or pictures which provide a visual feedback and input capability to the system. They are used to help reduce the learning curve, "facilitate user performance while reducing errors (Baecker and Buxton, 1987, p. 431)", and ease the pain of learning and remembering commands (Shu, 1988, p. 5).

Increased use of graphical representations and pictures is stimulated by the assumptions that pictures can convey more meaning than words, aid in understanding and remembering commands, provide an incentive to use a system, and provide fewer language barriers (Shu, 1989, pp. 7-8). Other styles typically used include menu systems, where system choices are presented to the user in a menu of alternatives, form filling dialogues, where the user fills in fields displayed in one or more forms on the screen, and graphical interaction, where the user creates and edits diagrams, drawings or images are also very effective styles which provide consistent information and intuitive manipulation of the system. (Baecker and Buxton, 1987, p. 427)

But the most intuitive dialogue is that of direct manipulation where the user manipulates the system through a series of buttons, toggles, and sliders which are graphic representations of the data. These types of dialogue provide the following functionality:

- "Novices can learn basic functionality quickly, usually through a demonstration by a more experienced user.

- Experts can work extremely rapidly to carry out a wide range of tasks, even defining new functions and features.
- Error messages are rarely needed.
- Users can immediately see if their actions are furthering their goals and if not they can simply change the direction of their activity.
- Users experience less anxiety because the system is comprehensible and because actions are so easily reversible.
- Users gain confidence and mastery because they initiate an action, feel in control and can predict system responses." (Schneiderman, 1987, pp. 202-203)

In a direct manipulation dialogue, the system provides continuous, dynamic, visible feedback of the object of interest.

The question now is "What is the best interaction style and the technique to integrate that style?" "The best interaction style is and will remain a complex function of the task, the users who are to carry out the task, the environment within which they will work, and the tools with which they are to do the job (Baecker and Buxton, 1987, pg 433)". Each of the dialogue styles mentioned has its own merit and provides varying degrees of functionality. The REFAB environment in which the user will operate will not be static and necessitates an interface which is adaptable to a collage of dialogue styles. Each phase differs depending on the information being presented and user needs for providing a more intuitive and flexible interface for constructing RESA game scenarios.

D. REFAB INTERFACE DESIGN

The successful implementation of any software system requires an amount of preliminary requirements analysis commensurate with the size of the project. In the case of the redesign of FORCE and BUILD, both users and designers are unsure as to what form the user interface should take. Additionally, the program function is already well defined, which in itself will drive many of the user interface requirements. In a case such as this, specific requirements analysis can be time consuming and wasteful. It is difficult to formally specify a user interface without the use of a prototype (Pressman, 1987, p. 53).

The prototyping process provides executable models depicting various attributes of the proposed system during the requirements analysis phase of software development. Prototyping allows the interface to be evaluated from the user level, providing feedback in the form of more refined requirements, and allowing the system to more fully meet the needs of the user once the system is designed, implemented and tested. The requirements analyst and user play an integral part in ensuring the requirements specifications meet the user needs. (Pressman, 1987, pp. 22-23)

Because the prototyping approach to requirements analysis and software development is being used, the software must be designed using structured modules and object-oriented programming techniques to ensure modifiability, extensibility, understandability, and transferability. Figure 3 depicts the sequence of events for the prototyping paradigm.

The steps required to implement a prototype are as follows:

- Define the overall objectives of the system specifying the known requirements and the "fuzzy" areas.
- Produce a quick design focusing on the aspects of the design visible to the user.
- Construct the prototype.
- User evaluates the prototype for refinement of requirements.

This cycle continues until the final version provides a model of the salient user critical features of the system and the developer has a better understanding of what is required.

Because REFAB is a redesign of an existing interface, the overall objectives of REFAB are already known and have been previously stated. The interface must provide the following BUILD functions:

- Create characteristics database files.
- Provide editing function to create or change characteristics in the database.
- Provide a workspace for editing of characteristics.

- Provide some controls to ensure that characteristics entered are complete and correct.
- Provide a capability to print database for review.

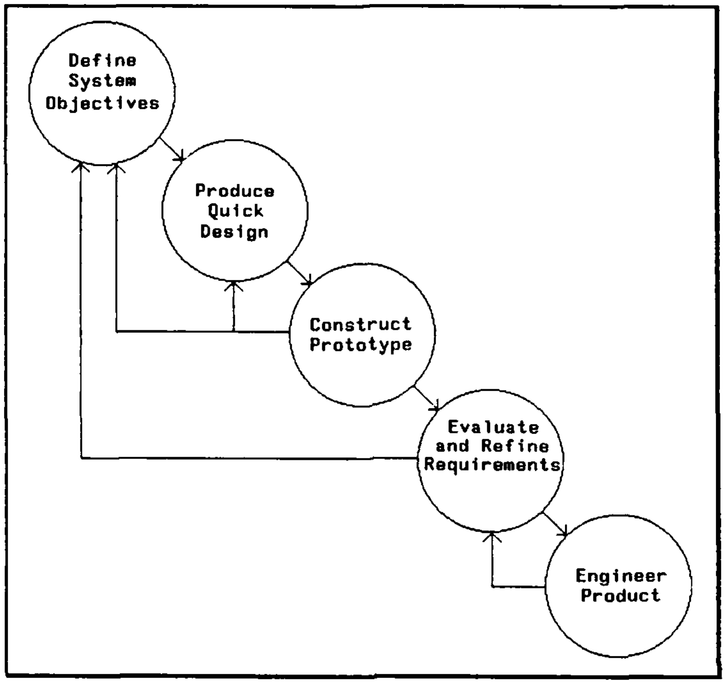

Figure 3 -Prototype Development Steps

REFAB must also provide the following FORCE functions:

- Provide mechanism to define scenario starting conditions.
- Verify force, sensor and weapon characteristics for scenario are contained in BUILD files.
- Provide mechanism for creating new scenario files, updating old files, or changing files after characteristics in BUILD are changed.
- Provide for printing data entered for review.
- Generate or delete force files.

The second phase in prototyping is identifying the form of a functional model. A very effective method is the use of an existing program or system which performs a similar function as the intended system. The existing system is then used as a base to sculpt and mold a functional model of the salient features critical to the system. The method used for REFAB was to modify an existing system, BATMAN and ROBIN.

III. SOFTWARE BASIS FOR FRONT-END REPLACEMENT

A. DESCRIPTION

The BATtle-MANagement Assessment (BATMAN) and the Raid Originator Bogie INgress (ROBIN) systems were developed by NPRDC, San Diego to test users against performance criteria based on allocating, deploying and managing surface, subsurface and air assets during simulated theater-level conflicts. These systems were designed for computer-based, single user performance testing using very low level modeling, but emphasized a very friendly user interface presented with animated, high resolution graphics.

These two systems were developed as sister systems. Multiple carrier-based, land-based, or amphibious assault ship-based task forces, can be designed in ROBIN and presented to the testee or user in BATMAN. ROBIN functions as a scenario generator allowing for the creation of hostile surface, subsurface and air platforms configured into Red force raids. The scenario generator also accepts the specification of platform tracks for the scenario, as well as chaff and communications jamming corridors. The scenario designer can then preview the Red movement for the scenario through a timed simulation. Additionally, the scenario designer can specify the number and type of Blue force tactical resources available to the testee for wargaming within BATMAN.

BATMAN provides the actual testing mechanism where the user must allocate, deploy and manage Blue surface, subsurface and air platforms against the Red force threat. The user is presented with up to three task forces based on mother platforms (carriers, land bases or amphibious assault ships). The user must allocate weapons to aircraft, aircraft to mother platforms, and surface and sub-surface platforms to the warfare theater to meet the hostile threat. The user must plan flight deck operation, assign combat air patrols (CAPs) of various types, schedule tankers for refueling, and preposition early warning aircraft. During game play, the user deploys fighters to intercept threat aircraft, and attack aircraft against hostile surface platforms. Once simulation ends, performance measures are presented to the user evaluating the users understanding and application of naval warfare strategy, tactics and decision-making performance. (Federico, 1989, pp. 1-3)

BATMAN and ROBIN's graphic interface is the key to the capabilities of this system and the ease with which users can manipulate the system. Both the user and the scenario designer are consistently presented with a theater level map of the current situations. All platforms are depicted as icons and commands are represented as buttons. The allocation of assets is as "simple" as clicking the mouse cursor on a platform and then clicking the cursor on the appropriate assets to be tagged for that platform.

B. SOFTWARE DESIGN

BATMAN and ROBIN are written in the C programming language and operate on the SUN 4/260 family of workstations under SUNOS 4.0, a UNIX-based operating system. They were designed using generic structures and object-oriented programming practices which aid in its flexibility and adaptability. This functionality was aided by the use of the Sun Visual/Integrated Environment for Workstations (SunView).

SunView is a software environment which supports interactive, graphics-based applications operating within windows. It is an object-oriented system where the main class of visual objects are windows. Other classes of objects include cursors, icons, menus, scrollbars, and panel-items. Windows can be either tiled using the sub-class frame or overlapping using the sub-classes canvas, panel, text sub-window, and TTY sub-window. BATMAN and ROBIN implement panels and canvas sub-windows exclusively.

Panels provide panel-items through which a user interacts with a system. These items include buttons, toggles, choices, messages, text, and sliders. Each item utilizes its own set of attributes to define its representation and location to the user. The library functions which define panel-items make use of variable argument lists so that the programmer may define the panel-items to suit the needs of the system. (Sun Microsystems, 1988, pp. 3-12)

The canvas sub-window is essentially a drawing window. It provides greater flexibility than a panel where it is not constrained to a fixed set of items. It uses lower level functions to access the window and must provide it's own routine to handle events.

BATMAN and ROBIN utilize a direct manipulation, graphic user interface where graphic objects or icons represent various platforms and functions. A theater level map is presented to the user on a SunView canvas. Platforms can be displayed, moved or tagged by moving and clicking the mouse cursor. The result is immediate feedback through real-time "visual differencing". The user can immediately visualize the results of all actions.

C. SOFTWARE COMPOSITION

BATMAN and ROBIN utilize five databases during the various game phases. The scenario database is an ascii text file representation of the platforms available for a specific scenario. The user database maintains a catalog of users permitted to operate the system. Both of these are very primitive, do not provide any functionality required by REFAB, and will not be discussed further.

The graphics database includes all icons and graphic representations of objects utilized in BATMAN and ROBIN. The graphics are all stored in standard Sun raster file format. Access to graphics database is afforded through the object definition database.

The object definition database is an ASCII text file list of all objects or platforms utilized in BATMAN and ROBIN. It is named ".defaults" and is initialized once the user enters the SunView environment. The file is a hierarchical list of option names and each line in the file consists of an object, parameter and value formatted as:

/object/parameter "value",

with each subsequent level separated by a slash. Option values are stored as strings and the string values can be retrieved as strings, integers, characters, booleans or enumerated types using the Sun library functions get_defaults_*. The format of this database will be discussed further in Chapter 4. (Sun Microsystems, 1988, pp. 145-154)

The fifth database is the database of World Database II maps from the Applied Physics Laboratory at Johns Hopkins University. The maps are rendered by an orthogonal projection of the globe onto a horizontal plane at a specified range and latitude-longitude (lat-long) coordinate. The maps can be drawn with or without political borders and lat-long lines. The maps are accessed through a package of library routines which draw the maps on a SunView pixrect or picture rectangle.

Additionally, the software includes seven other packages. Because REFAB is only concerned with database definition and scenario design, the BATMAN simulation and performance measures packages were not considered. The initialization and control package initializes the window environment, scenario environment, and scenario data structures. The loadout package handles weapons loadout for Blue air platforms and creates the tactical situation map. The deployment package provides access to the scenario situation map for the deployment of Blue tactical assets and the setting of initial alert levels. The Robin package implements the scenario generator for creating and editing Red force raids and assigning platforms to Blue task forces. The last package is a set of tools which provide mechanisms for creating and manipulating three key window objects within BATMAN and ROBIN.[Federico, 1989,pp. 16-50]

The Panel_win routines create and manipulate a data structure based on the Sunview Panel, but with "additional information to provide added functionality" (Federico, 1989, p. 48). This package allows for the creation of a variable number of generic Panel_items displayed as icons and is utilized extensively for weapons and aircraft loadout. Additionally, it maintains the screen coordinates for the visible and hidden rectangles. The abstract definition is provided in Figure 4.

```
PANEL_WIN
            ::=   Sunview Panel
            +     visible rectangle
            +     hidden rectangle
            +     SunView Panel_items[]
            +     icon pixrect[]
            +     x-location of icon[]
            +     y-location of icon[]
```

Figure 4 -Panel_win Structure

The Canvas_win is another package which creates and manipulates a data structure based on a SunView object, the Canvas. It also provides the additional information of visible and hidden rectangles as well as a mechanism where the Canvas can act like a Panel and provide access to pseudo-buttons. The abstract definition for the Canvas_win is shown in Figure 5.

Finally, BATMAN and ROBIN utilize an instance of the Panel_win for displaying popup messages, which contain buttons, messages or text items. One private Panel_win is utilized to display the popup messages and is cleared between each use.

BATMAN and ROBIN incorporate the windowing functionality provided by SunView, the graphics representation provided by a bit-mapped display, and the use of menus and direct manipulation afforded by an event-driven interrupt handler.

```
CANVAS_WIN
                ::=   Sunview Canvas
                 +    visible rectangle
                 +    hidden rectangle
                 +    psuedo Canvas_items[]
                 +    icon pixrect[]
                 +    x-location of icon[]
                 +    y-location of icon[]
```

Figure 5 -Canvas_win structure

The combination of RESA's high-level models with BATMAN and ROBIN's graphic interface appears to be a likely combination.

IV. PROTOTYPE IMPLEMENTATION

The method of development utilized to prototype REFAB was to extract all applicable source modules from BATMAN and ROBIN. This approach has provided an economical, quick implementation of a full scale visual interface. BATMAN and ROBIN provided a good basis for this development effort because of the similar nature in program function to RESA.

Our major design criteria was to maintain compatibility with RESA. The first draft prototype had to provide access to the same command functionality as provided within FORCE and BUILD. Users already familiar with FORCE and BUILD must be comfortable in front of the new interface to be able to manipulate the system with little or no documentation. REFAB is divided into a build phase and a force phase mirroring the BUILD and FORCE programs of RESA.

A. REFAB ENVIRONMENT

REFAB was developed and operates on a Sun Microsystems SPARC System-370 workstation which utilizes the Sun 4300 CPU. These workstations provide a powerful graphics environment through the use of standard (1142 x 870 pixel) or increased resolution (1600 x 1280 pixel) monochrome, color or gray-scale monitors, the choice of an 8 or 24 bit plane graphics frame buffer, and use of the SunView windowing environment. The use of the Sun workstation was driven by the use of BATMAN and ROBIN as the prototype. Utilization of the Sun SPARC station was also predicated on the fact that the map routines to access the World Database II maps are in an object library compiled for the SPARC environment.

REFAB is written in "C" due to both the use of BATMAN and ROBIN, and its implementation on the Sun workstation. "C" is very closely associated with the UNIX operating system on the Sun and eases the interfacing with the SunView environment. It provides the power and structure of high level languages with the absence of restrictions. The use of SunView affords the opportunity to utilize programming techniques indicative of object-oriented languages.

Object-oriented programming is more of a technique for programming in a language than it is using an object-oriented language. It is a "paradigm for writing good programs"(Stroustrup,1988, p. 10) while object-oriented languages provide constructs which support the object-oriented style. This style is one which provides data abstraction, data hiding and is modular.

Programming in modules centralizes data of a specific type and places it under the control of a type manager. The modules define data as a type, allocate memory for that type and return not the data but an identifier or pointer to that type. This modularity is provided throughout SunView where classes of objects (panels, panel_items, canvases) are defined by a set of library routines and all that is returned to the programmer is an identifier or handle. The programmer knows the types and the operations associated with the identifier and that is all that is required to manipulate the object.

The standard practice in C is to partition programs into modules which hide data. This is done by defining a separate include file containing only the declarations required for external use. All other data is declared static or private to that module thereby achieving another degree of modularity. In addition to the language choice and environment restriction, extensive modifications of the BATMAN and ROBIN code had to be made.

B. MODIFICATIONS TO PACKAGES

REFAB is intended to be a stand-alone interactive editor and as such does not require any of the simulation function provided by BATMAN. This includes the use of the simulation engine, and the detection, movement and update models.

The requirement for a user database was also irrelevant because of the multi-user nature of RESA. All performance measures within the game are group oriented instead of individual. As such, there is no "login" requirement for REFAB as in BATMAN and ROBIN.

REFAB is not a straight translation from BATMAN and ROBIN; the functions for configuring Red raids and Blue task forces are divided between BATMAN and ROBIN. For REFAB, the configuration and loadout were centralized into the force phase as it is currently in FORCE. Table 1 lists the packages deleted from BATMAN and ROBIN.

Table 1 - BATMAN AND ROBIN PACKAGES DELETED

C Modules and Header Files Deleted		
alert.c,.h	detect.c,.h	engine.c,.h
jtids_antenna.c,.h	jtids_antenna_load.c,.h	jtids_conn.c,.h
jtids_hooks.c,.h	jtids_network.c,.h	number_pad.c,.h
plat_detect_funcs.c,.h	plat_list_funcs.c,.h	plat_update_funcs.c,.h
playback.c,.h	robin_defcon.c,.h	robin_manage.c,.h
show_colors.c,.h	stats.c,.h	stats_compute_funcs.c,.h
stats_notify.c,.h	stats_update_funcs.c,.h	stats_verfiy.c,.h
status.c,.h	user_db_access.c,.h	user_funcs.c,.h

REFAB uses the Panel_win, Canvas_win, and Popup_panel packages as a basis of presentation for the user. The Panel_win and Canvas_win packages were used without modification. The Popup_panel was modified to allow the use of multiple text items. A single masked text item was provided with BATMAN and ROBIN as a password tool. The designer can now provide popup panels which present multiple textual input fields.

C. WINDOW LAYOUT

Central to the REFAB interface is the use of windows. The build function within REFAB provides a screen consisting of three horizontal SunView panels as depicted in Figure 6. The bottom panel is a status panel which provides current system settings of the name of the open database and the name of the open characteristics file. Additionally, a choice item is provided to alter the step increment of sliders in the workspace panel. Additional status features will be added in the future as the interface expands.

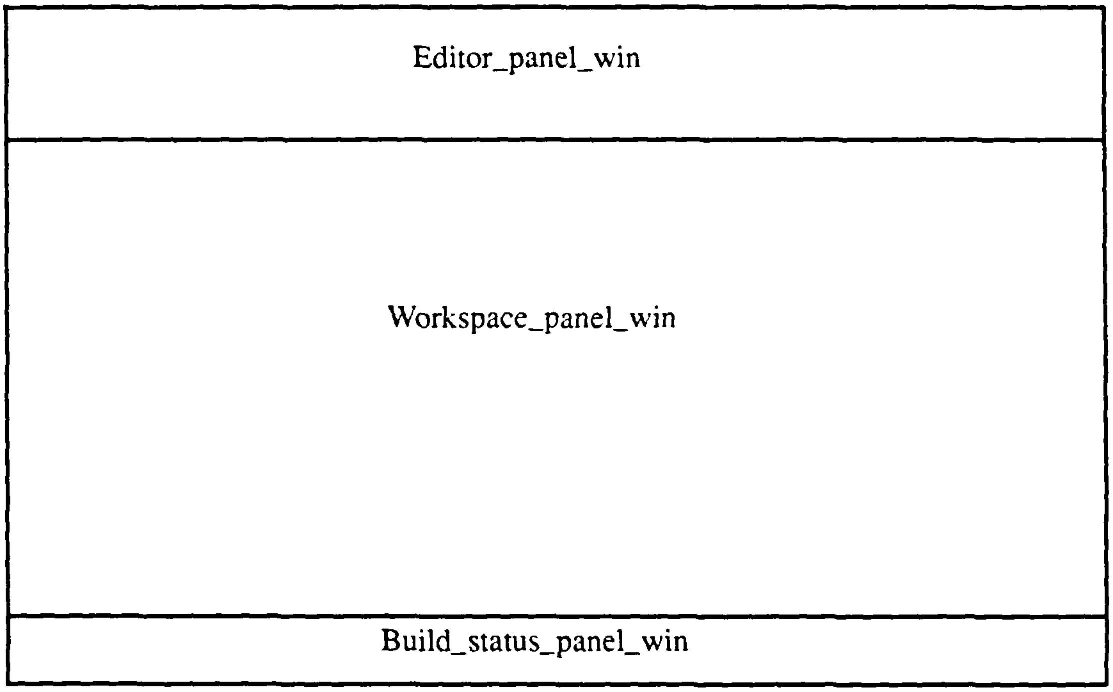

Figure 6 -Build Function Screen Layout

The top panel provides buttons representing the BUILD program control orders **PRINT** and **BYE** as shown in Figure 7. The **WRITE** order was made obsolete by the use of a direct manipulation interface and as such was deleted. To preserve the idea of a "modeless" environment, a button was implemented to access a menu of object access orders. The **OPEN** button (currently not shown) presents a menu of objects when pressed. Once an

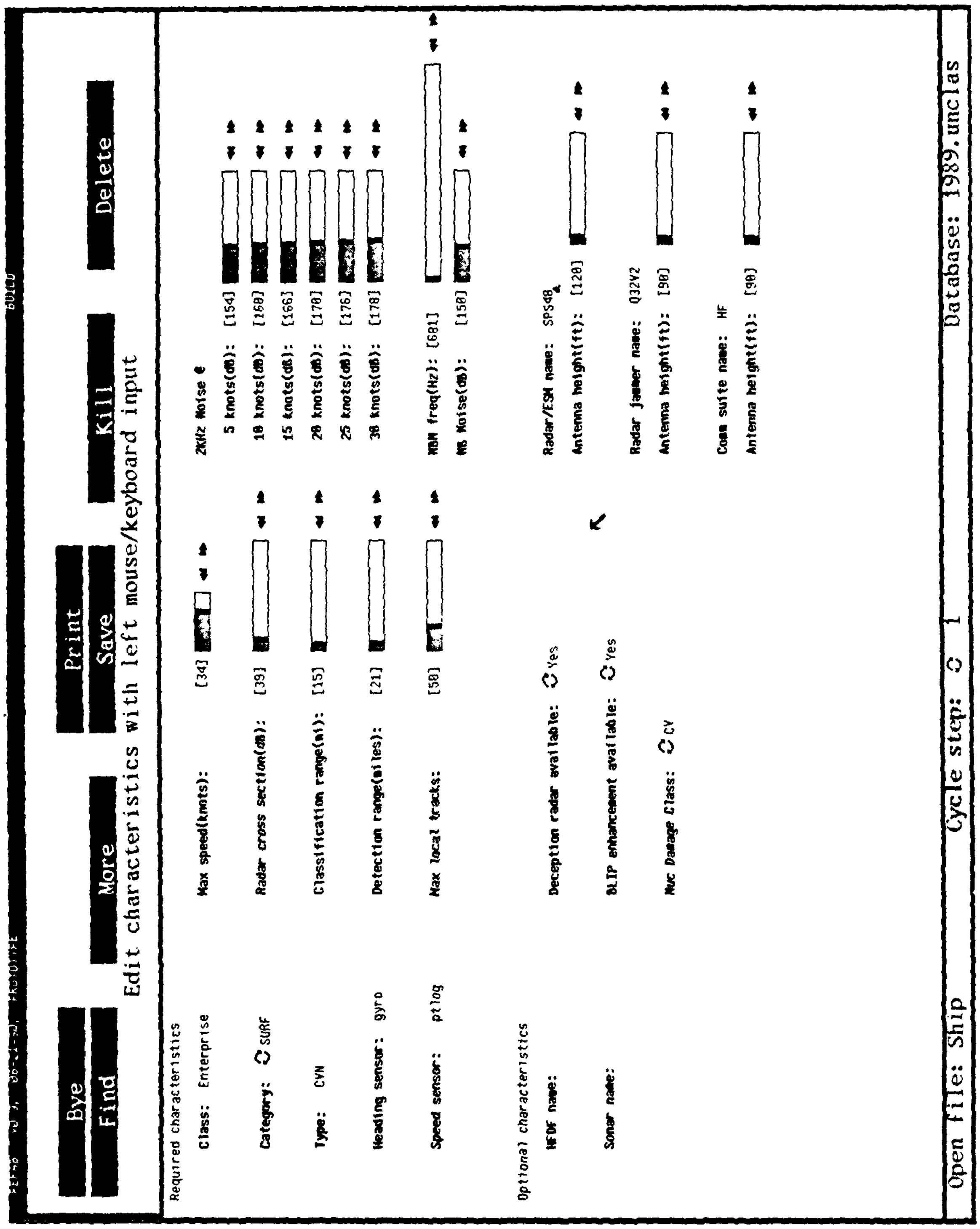

Figure 7 -Build Function Display with Workspace Open

object is selected, the button is cleared from the display. This protects the user from trying to open two characteristics files at once. Additionally, a second row of buttons representing the characteristics control orders is displayed giving the user the functionality of locating objects and editing their characteristics. Once the user is through editing an open file, REFAB utilizes the same command as BUILD, BYE. This toggles the OPEN button back on, the characteristics control orders off, and updates the status panel. Exiting REFABs build function is accomplished in the same manner as BUILD by using the BYE button.

The middle panel, the workspace, provides a form-filling dialogue of text-items, sliders, and choice items for editing an open characteristics file. The layout of the workspace panel is characteristics file dependent, but at this time only the ship characteristics can be displayed. This dialogue affords the user a more insightful presentation of the required and optional characteristics, and a graphic depiction of the range or choice of values. Choice items are used to limit selections to specific names, and sliders are used to present the user with a graphic depiction of the range of numeric values, a function not provided within the RESA system.

The force function within REFAB provides a screen consisting of a SunView canvas for displaying a theater level map, and three panels which border the canvas top, bottom and right side. Figure 8 illustrates the layout for this function. Map canvas provides a two dimensional theater-level map using the World Database II map package from the Applied Physics Lab at Johns Hopkins. The map package is a good visual cue for the tactical state of deployment of objects. Although this is not new to RESA, this functionality has never been afforded to the pregame processes. Users currently have to guess locations or use previously generated scenarios to define scenario initial conditions. Manipulation of the canvas is accomplished through the use of the tools panel. The final prototype layout for the force function is depicted in Figure 9.

The tools panel provides a series of buttons labeled with picture images, which aid in the manipulation of the map canvas. This panel currently provides buttons to remove objects form the canvas, a zoom function, and a button to move and change the theater displayed.

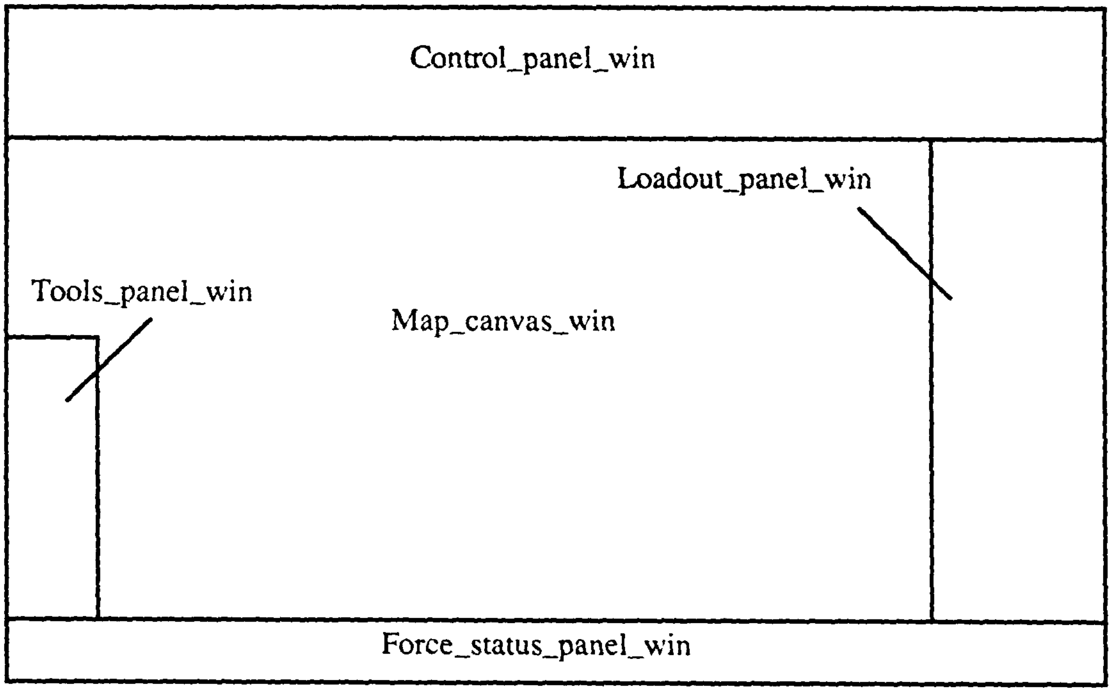

Figure 8 -Force Function Screen Layout

As in the build function, the bottom panel is again the status panel and displays the currently opened database and scenario files, and positioning information (latitude, longitude, theater) of task forces. The right panel currently is utilized as a loadout work area for allocating assets to task forces, both RED and BLUE. Future enhancements will utilize this panel as a general work area for graphically defining the other force scenario parameters for wargame initialization.

The top panel is again utilized as a direct manipulation platform of buttons for accessing the force program and scenario control orders. The control order ENTER is utilized to enter

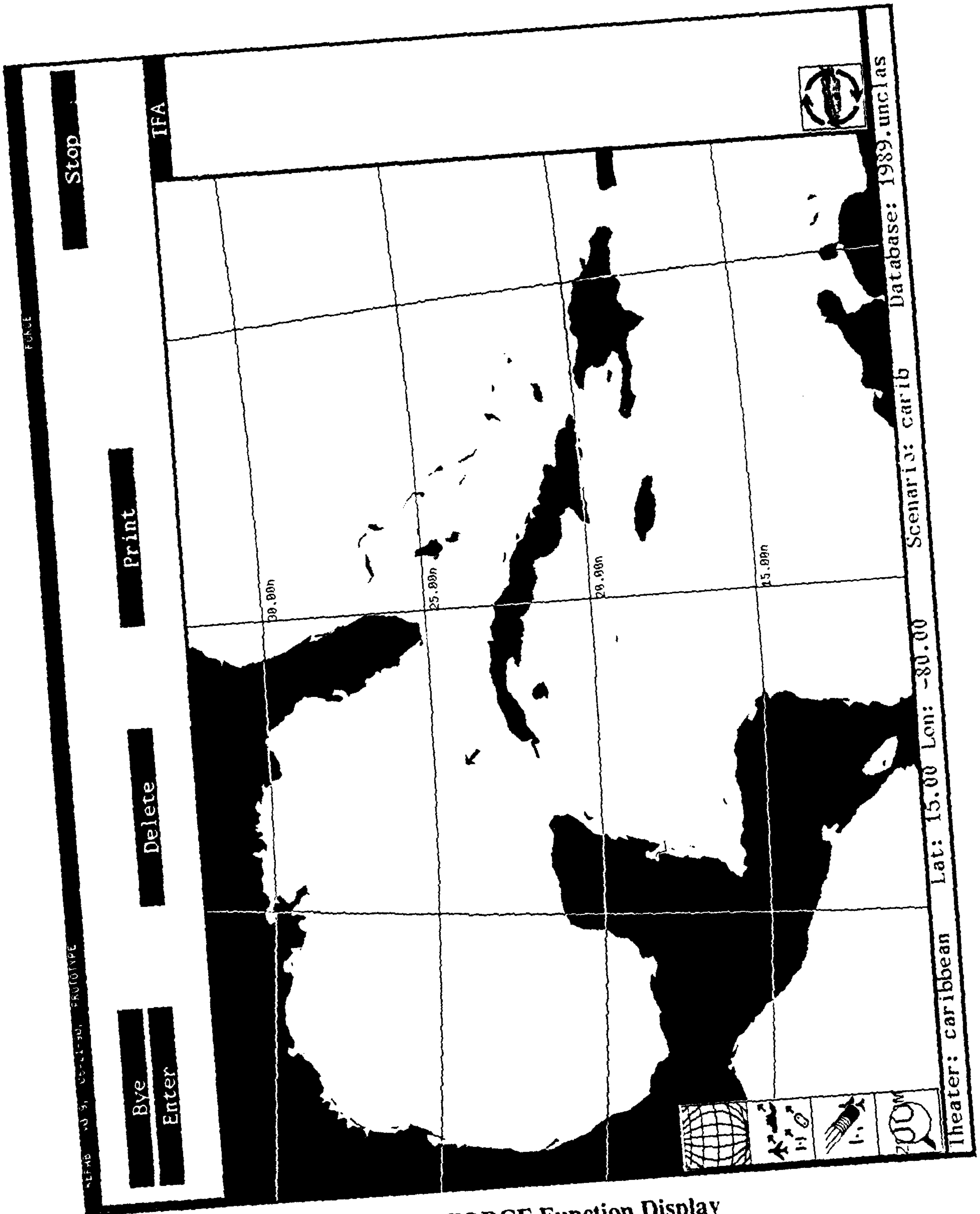

Figure 9 -FORCE Function Display

instances of objects for the initial conditions of the wargame. Objects will be displayed on the loadout panel, dragged on the map canvas, and positioned in the warfare environment. The map canvas will be utilized in the same manner as a drawing utility where the user can perform such graphic functions as "drawing" paths for ships, tracks for air, corridors for communications jamming, and ship communications paths.

D. MODIFICATIONS OF DATA STRUCTURES

Central to BATMAN and ROBIN is a solitary data structure for maintaining control of the window objects, icons, simulation engine nodes, painting operations, fonts, and Blue and Red platform force lists. This data structure is shown in Appendix C. It is apparent that this structure is a depository of many global variables. Because FORCE and BUILD are essentially separate functions, no requirement exists to tie these functions together via one data structure. Global variables and defined SunView objects in REFAB are kept separated from the scenario data structures and database structures. Figure 10 identifies the structure used for ship objects in REFAB. Note that the variables heading_sensor, speed_sensor, radar, jammer, sonar, commsuite, weapon_type, and missile_type are all pointers to other structures defined by their respective characteristics file.

Another aspect of change to data structures was the use of the defaults database. BATMAN and ROBIN use the defaults database exclusively for object characteristics. But because the models in RESA are so much more extensive, new data structures had to be developed. The ".defaults" file is a good tool for storing system parameters that are used once to initial the system and allow the user to customize the interface. However, the continuous disk access required to read the database each time an object is referenced produces far too much overhead and is very inefficient. No attempt was made to perform a straight translation from the Fortran data structures of RESA which consists exclusively of arrays. "C" provides a very elegant and efficient method of linking data through the use of record structures and pointers. REFAB uses a list of records for each characteristics file and a list of records with pointers to the characteristics structures for scenario generation. Figure 9 depicts this concept for the ship objects. Fields have been added to the REFAB structures to access SunView pixrects or icons. Once a named object is initialized, its name

```
typedef struct ship_head {
            char              *class;
            char              *cat;
            char              *type;
            Pixrect           *small_icon;
            int                maxspeed;
            int                radar_xsect;
            sensor_rec        *heading_sensor;
            sensor_rec        *speed_sensor;
            int                class_range;
            int                detect_range;
            int                local_tracks;
            int                BBN[6];
            int                BBN3[6];
            int                NBN_freq[6];
            int                NBN[6];
            int                deception;
            int                Blip_enhance;
            radar_rec         *radar[8];
            int                radar_antenna_ht[8];
            jammer_rec        *jammer;
            int                jammer_antenna_ht[8];
            sonar_rec         *sonar[6];
            commsuite_rec     *commsuite[12];
            int                commsuite_antenna_ht[10];
            weapon_type       *weapon[30];
            missile_type      *missile[30];
            int                nuc_damage_class;
}
```

Figure 10 -REFAB Ship Structure

or class can be looked up in the defaults database and the picture can be drawn on a SunView object. In this way, icons can be easily added or replaced from outside the system.

The method of implementation of pictures as icons typically in SunView is to define a button with a picture image instead of a label. Then when the icon is selected, a defined procedure is called to handle this event just as if a button had been pressed. This is depicted in the tools panel in Figure 7.

To maintain the list of structures, extensive use is required of the list manager routines provided with BATMAN and ROBIN. These routines provide a generic mechanism for manipulating doubly linked lists.

E. ADDITIONAL FUNCTIONS

In addition to the display descriptions presented, the following design decisions were made to present a consistent interface providing informative feedback and reducing short term memory load. In keeping with the standard Sun usage, the left mouse is utilized for "picking" objects, and the right mouse is used for menu access where applicable. Because the middle mouse has no defined function and is often system dependent, REFAB uses it for context sensitive help. Utilizing the functionality of popup panels acquired from BATMAN and ROBIN, REFAB provides help panels for buttons and panels. Currently, only the **OPEN** and **ENTER** buttons have help panels implemented as they are the only orders currently tied to event procedures to handle user input.

SunView provides an event-driven interrupt handler for grabbing all user input for the system. This capability allows the implementation of context sensitive help where each panel and panel_item defines its own event handler which in turn displays related help screens.

The event handler allows the system to identify which mouse buttons are pressed, whether they are up or down, cursor position in screen coordinates and whether cursor is stationary or being dragged. All of these capabilities allow the user to obtain different feedback depending on the actions performed and preserving the impression that the user is in control.

System messages and errors are displayed with the same popup utility as the help messages. In this way, error messages are not depicted as "finger pointing"; they are the same as system wait messages and help message requested by the user. The design of the popups naturally draws user attention and are turned off either by the system when the system is through processing or by the user when he wishes to continue.

In addition to the user-requested help screens, both the build and force functions have a prompt item in the control panel which is utilized to guide the user through the system and minimize the requirement for system documentation. Prompts are utilized to ensure all required parameters for an order are obtained from the user. In this way, difficult tasks can be made easy by breaking them down into smaller easier to manage tasks. The user then feels more in control and does not become frustrated with the system when he doesn't know what to do next.

Though the use of color is considered restrictive and leads to non-portable applications, the use of color is applied in REFAB to make images more meaningful and guide the user to appropriate information. Grouping related information through color coding retains the usefulness and readability of the screen. Color is used in REFAB for the buttons and utilizes the "traffic light" metaphor familiar to most users. Potentially dangerous commands, such as **KILL** and **STOP**, are identified by a red button (Stop), commands which may require some type of reversal action, such as **BYE**, are depicted in yellow (Caution) and commands for normal system operation are shown in green (Go). In this way, the dialog yields closure by associating a familiar color scheme with the feedback expected from buttons.

In REFAB, each button has its own menu associated with it. Currently these menus are static because of the state of the database translation. However in the future implementations, menus of object names and classes will have to be generated dynamically depending on the database in use and the state of the scenario generation. Menus are designed to yield closure where actions expected first are listed at the top and actions expected last are listed at the bottom. For example, users are expected to define ships before aircraft and boats, and as such "Ships" appears at the top of the menu above "Air" and "Boat" in the **ENTER** order.

BATMAN and ROBIN provides another private popup panel, a number pad, for identifying the number of objects required in the scenario. The number pad is a good graphic application but does not provide the simplicity required for REFAB. The BUILD database consists of numerous integer variables which have varying ranges and default values. If the user has to look up "reasonable" values for these variables every time an objects characteristics are changed, then REFAB is not supporting the user. The interface should provide the user with that information. REFAB utilizes a slider module which defines a generic slider with a minimum and maximum value, a label, and two arrow buttons to add in moving the slider value in fixed increments. The increment steps can be set through the status panel. In this way the user is presented with a reasonable scale of values and not a mechanism for arbitrary choices. The slider mechanism can be used in the force phase to display reasonable loads for aircraft as well as air wing configurations for carriers or shore bases.

V. RECOMMENDATIONS

A. FUTURE RESEARCH

The overall goal of this research was to develop a first draft prototype replacement of FORCE and BUILD outlining the types of graphical depiction possible. Because this is a first draft, REFAB has certain limitations which need to be addressed with future research.

Currently, the database is designed as a linked list utilizing the list manager. For a database as large as required for BUILD, this structure is inappropriate and the list manager will have to be redesigned for structures typically used to store and access databases, such as binary trees, B-trees, or AVL trees.

Because of the limited nature of the database, some of the menus have been implemented as static structures. The menus for listing of database object names and classes is also currently a static structure. To allow the use of a number of different databases, REFAB will have to be redesigned to allow for dynamic building of the menus.

The World Database II maps and associated object libraries can currently only be utilized on a Sun-4 SPARC station with a color graphics display. If REFAB is to be portable to other Sun architectures or translatable to other windowing environments, then the source will have to be obtained for the map libraries. Additionally, the functionality of the map package is limited to the routines used in BATMAN and ROBIN. The object modules provide no reference as to the description or parameters required of other routines.

B. CONCLUSIONS

The development of REFAB has demonstrated the feasibility of enhancing the interface of the pregame processes of RESA. This first phase of the prototype design has provided an effective medium through which requirements analysis can be implemented and encourages the identification of requirements modifications and enhancements. The designer and user can now refine the requirements until a functional model is produced which supports all of the salient, critical features of FORCE and BUILD. Through the use of the Sun workstation and bit-mapped graphics, REFAB presents an interface which is intuitive and will require a minimal learning curve. The object-oriented approach has produced a program which is modifiable, general, supports abstraction and is extensible. The use of UNIX operating system also provides an affective vehicle for networking and process sharing which will be essential in integrating REFAB with RESA. REFAB has great potential of providing users with an interface for RESA which will yield a more satisfying and rewarding wargaming session.

Through the use of consistency, users familiar with RESA are presented with a graphic portrait of the available orders in FORCE and BUILD in a consistent layout. The top of the screen provides a panel for user orders, the bottom a panel for current system status items. The name of the database is even presented in the same place. Both orders panels also provide a system prompt area. The buttons are of a consistent size and use a standard coloring scheme. The buttons are also consistent with RESA by utilizing the same naming conventions for orders and characteristics. The characteristics are also consistent with real world naming and values. Except for the system prompt which is designed to unobtrusive, all messages are provided in a consistent format by use of the popup panel.

The implementation of the style of dialogue presented with the REFAB interface could significantly reduce user training requirements and provide users with a more productive wargaming experience emphasizing decision-making and not keyboard entry corrections. Additionally, an interface such as REFAB could reduce the need for operator personnel allowing users easier interaction with the system, and reduce resource costs utilizing a single bit-mapped display to provide all information through windows instead of three different displays. It is hoped that his preliminary design will be reviewed by the RESA facility at NOSC as a benefit to RESA and will aid in the current efforts of defining the direction of wargaming in the future.

APPENDIX A.

RESA PLATFORM AND SYSTEM CHARACTERISTICS

The BUILD database is configured from fifteen files. The database name consists of five alphanumeric characters starting with an alphabetic character. This character string replaces the XXXXX in the fifteen files listed in this appendix. Each characteristic tuple is defined by a field name, the number of instances allowed, a range of values, and a description of the characteristic. A minus(-) sign preceding the number of instances indicates an optional characteristic.

Object: Aircraft Filename: XXXXXC.AIR

Field	Number	Range of Values	Description
NAME	1	alphanumeric	Name of aircraft
CAT	1	JET|PROP|HELO	Aircraft category (defined type)
VMAX	1	0:3000	Maximum speed (knots)
CLIMB	1	0:10000	Maximum rate of climb (feet/min)
AMAX	1	0:150000	Maximum obtainable altitude (feet)
XSECT	1	0:20	Radar cross section (dB)
VCRU	1	0:1000	Optimal speed for maximum range (knots)
JP	1	0:300000	Amount of aviation fuel (lbs)
LDELY	1	0:10	Launch delay for appropriate aircraft (min)
HDG	1	alphanumeric	Name of heading sensor
SPD	1	alphanumeric	Name of speed sensor
MNFAL	1	0:1000	Mean time before failure (min)
MNMNT	1	0:1000	Mean time for unscheduled maintenance (min)
SDMNT	1	0:1000	Standard deviation unscheduled maintenance (min)
MNRPR	1	0:1000	Mean time to repair (min)
MAINT	1	0:1000	Flight time between scheduled maintenance (min)
RDELY	1	0:100	Routine servicing and refueling delay time (min)
ODELY	1	0:100	Change-of-ordnance delay (min)
PBLCH	1	0:100	Prob of a sucessful launch (%)
PBREC	1	0:100	Prob of a sucessful recovery (%)
PBFAL	1	0:100	Prob of a system failure (%)
SYSFL	1	0:100	Prob of a sys failure while in flight (%)
DTRNG	1	0:500	Range in which the platform can be detected (miles)
CLRNG	1	0:500	Range in which the platform can be classified (miles)
TRKS	1	0:100	Number of local tracks (qty)
LOTSP	1	0:4095	Loiter speed for station (knots)
VECSP	1	0:1023	Vector speed for engaging (knots)
HFDF	- 1	alphanumeric	Name of HFDF
FLIR	- 1	alphanumeric	Name of Foward-Looking Infra Red (FLIR)
NAV	- 1	alphanumeric	Name of navigational system
RDESM	- 6	alphanumeric	Name of sensor(s) carried
JAMMR	- 1	alphanumeric	Name of jammer carried
COMMS	-12	alphanumeric	Name of communication suite(s) carried
COMMJ	- 4	alphanumeric	Name of communications jammer(s) carried
SONAR	- 6	alphanumeric	Name of sonar(s) carried
NDCLS	- 1	1:3	Nuclear Damage Class (1-B 2-F 3-H)
DNWND	- 1	YES|NO	Can land downwind? (YES/NO)
ISAR	- 1	YES|NO	Ship-ID @ min (80 nmi horizon)
ZOOM	- 1	1:4	Visual classification range multiplier

Defined type: HELO

Field	Number	Range of Values	Description
LBSKT	1	0:100	Pounds per knot
LOITF	1	0:4095	Loiter fuel (preprx)
HOVER	1	0:16000	Hover factor for max range

Defined type: PROP

Field	Number	Range of Values	Description
FLXPT	1	0:100	Exponential factor for fuel
FLPRX	1	0:4095	Prefix factor for max range
LOTFL	1	0:16000	Loiter factor for max range

Defined type: JET

Field	Number	Range of Values	Description
FLXPT	1	0:100	Exponential factor for fuel
FLPRX	1	0:4095	Prefix factor for max range
LOTFL	1	0:160000	Loiter factor for max range

Object: Communications Pairs Filename: XXXXXC.CMP

Field	Number	Range of Values	Description
NAME	1	alphanumeric	Name of Communications pair
COMMS	1	alphanumeric	Name of commsuite for sending
COMMR	1	alphanumeric	Name of commsuite for receiving at next node
LOS	1	YES I NO	Limited to line of sight?
MU	1	1:2000	Capacity of comm circuit (messages/hr)
RHO	1	0:100	Path utilization factor (%)
LIFE	1	1:500	max time to deliver a msg to COMMR (minutes)
RANGE	1	0:15000 I ACOUS	Maximum range for communication (miles)
FREQ	1	3:3000000	Frequency of commpair (kHz)
CRYPT	1	YES I NO	Is it encrypted?
JMRSN	1	5:50	Sensitive to jamming (qty)
SPEED	- 2	1:30	Two max speeds: in-layer below-layer (knots)
WIRE	- 2	SEND I RECR	Define if SEND or RCVR of comm needs a wire
WBUOY	- 2	SEND I RECR	Define if SEND or RCVR of comm needs a Wbuoy
MAST	- 2	SEND I RECR	Define if SEND or RCVR of comm needs a mast
MISSN	- 1	YES I NO	A/C mission overide for relay
ARC	- 2	SEND I RECR	Range of sender off receivers course
RCVRC	- 1	0:180	Receiver course off N/S where comm possible (degrees)

Object: Communications Jammers Filename: XXXXXC.CMJ

Field	Number	Range of Values	Description
NAME	1	alphanumeric	Name of comm jammer
RANGE	1	0:500	Max range of comm jammer
POWER	1	0:10000	Output power (Watts)
GAIN	1	0:20	Antennagain (dB)
COVER	1	0:100	Bandwidth (%)
FREQ	10	1:10000	Center frequencies and (MHz)
		1:50000	Bandwidths (Hz)
DELAY	1	0:10	Delay factor and (usec)
		1:50000	Bandwidth (Hz)
MNFAL	- 1	0:10000	Mean time between failure (min)
MNRPR	- 1	0:10000	Mean time to repair (min)

Object: Communications Buoys Filename: XXXXXC.CMB

Field	Number	Range of Values	Description
NAME	1	alphanumeric	Name of the commbuoy
LIFE	1	1:200	Buoy life when deployed (minutes)
MULTI	1	YES|NO	Single or multiple message buoy?
COMM1	1	alphanumeric	Name of 1st communication suite(s) carried
COMM2	1	alphanumeric	Name of 2nd communication suite(s) carried
MNFAL	- 1	0:10000	Mean time between failure (min)
MNRPR	- 1	0:10000	Mean time to repair (min)

Object: Communications Suites Filename:XXXXXC.CMS

Field	Number	Range of Values	Description
NAME	1	alphanumeric	Name of commsuite
BAND	1	LAND | LF | HF HFLRI | UHF | VHF VLF | ACOUS	Frequency band
TYPE	1	SEND | RECV | BOTH	Type of commsuite
FREQ	1	1:1000	Center frequency (MHz) and
		1:3000000	Bandwidth of commsuite
XPOWR	1	1:1000	Output power of transmitter (Watts)
XGAIN	1	1:10	Transmitter antenns gain (dB)
RGAIN	1	1:10	Receiver antenna gain (dB)
HOPPR	1	YES | NO	Does equipment employ frequency hopping?
PULSE	1		Signal pulse length (msec)
MNFAL	1	0:10000	Mean time between failures (min)
MNRPR	1	0:10000	Mean time to repair (min)

Object: Cruise Missiles Filename:XXXXXC.CRU

Field	Number	Range of Values	Description
NAME	1	alphanumeric	Name of missile
VCRU	1	0:2500	Average speed (mph)
FLDUR	1	0:300	Maximum flight duration (minutes)
EPU	1	0:100	TNT equivalent payload (hundreds of lbs)
RATE	1	0:10	Firing rate (missiles/min)
ACRU	1	0:100000	Altitude at which a missile travels (feet)
XSECT	1	0:20	Radar cross section (dB)
SEEKR	1	RADAR I RDESM	Seeker type carried
RADAR	1	alphanumeric	Type of radar carried
SKRNG	1	0:200	Maximum detection range (nautical miles)
ARC	1	0:90	Max view arc of missile sensors (degrees)
PROTL	1	NEAR I FAR	Target select protocol
PHD	1	0:100	Deception effectivity (%)
PHB	1	0:100	BLIP effectivity (%)
PHC	1	0:100	CHAFF effectivity (%)
PHJ	1	0:100	Jamming effectivity (%)
PL	1	0:100	Probability of a launch (%)
PH	1	0:100	Probability of a hit (%)
DEPTH	1	0:100	Maximum launch depth of a missile (feet)
HDG	1	alphanumeric	Name of heading sensor
SPD	1	alphanumeric	Name of speed sensor
NAV	-1	alphanumeric	Name of navigation aid
EMIT	-10	alphanumeric	Name of emitters recognized
TSHIP	-1	YES I NO	Can this missile attack ships/subs?
TAIRC	-1	YES I NO	Can this missile attack aircraft?
TCRUZ	-1	YES I NO	Can missile attack cruise missiles?
BURST	-1	AIR I SURF I SUBSU	Burst type (Nuclear)
BDPTH	-1	SHALL I DEEP	Depth of subsurface burst (Nuclear)
YIELD	-1	0:32767	Yield of Nuclear burst (kt)

Object: Radar/ESM Jammers Filename: XXXXXC.JAM

Field	Number	Range of Values	Description
NAME	1	alphanumeric	Name of Jammer
POWER	1	0:25000	Power(dB/watt)
RANGE	1	0:500	Maximum effective range (nautical miles)
GAIN	1	0:25000	Antenna Gain (dB)
FREQ	- 10	1:32393	Center freq and bdwth of jammer (MHz)
MNFAL	- 1	0:300	Mean time between failures (minutes)
MNRPR	- 1	0:300	Mean time to repair (minutes)

Object: Navigation Aids Filename: XXXXXC.NAV

Field	Number	Range of Values	Description
NAME	1	alphanumeric	Name of navigational device
CAT	1	HDG | SPD | DR OMEGA | TRANS | OTRAN	Type of navigational device (defined type)
MNFAL	- 1	0:10000	Mean time between failures (minutes)
MNRPR	- 1	0:10000	Mean time to repair (minutes)

Defined type: HDG

Field	Number	Range of Values	Description
HERR	1	0:1000	Std dev of heading error (deg/100)

Defined type: SPD

Field	Number	Range of Values	Description
SERR	1	0:5000	Std dev of Speed sensor error (knots/100)

Defined type: DR, OMEGA, TRANS, OTRAN

Field	Number	Range of Values	Description
LTRSD	1	0:100	Latitude bias error rate (yds)
LGRSD	1	0:100	Longitude bias error rate (yds)
LTHFX	1	0:100000	Fixed human error comp of lat (yds)
LTHSD	1	0:20000	Std dev human error comp of lat (yds)
LGHFX	1	0:100000	Fixed human error comp of long (yds)
LGHSD	1	0:20000	Std dev human error comp of long (yds)

Object: Surveillance Satellites Filename: XXXXXC.SAT

Field	Number	Range of Values	Description
TYPE	1	alphanumeric	Type of satellite
SWATH	1	1:3000	Radar swath width (miles)
SENSR	1	alphanumeric	Name of sensor carried
DUTY	1	1:90	Duty cycle per orbit (degrees)
HORBW	1	0:359	Horizontal beamwidth (degrees)
VERBW	1	0:359	Vertical beamwidth (degrees)

Object: Radar/ESM Systems Filename: XXXXXC.SEN

Field	Number	Range of Values	Description
NAME	1	alphanumeric	Name of sensors
CAT	1	RADAR | ESM	Type of sensors (defined type)
BDWTH	1	1:20000	Recevier bandwidth (MHz)
NSFIG	1	0:2000	Radar noise figure (dB)
GAIN	1	1:2000	Antenna gain (dB)

Defined type: RADAR

Field	Number	Range of Values	Description
SRATE	1	0:4320	Scan rate (scans/min)
FREQ	1	1:20000	Sensor operating frequency (MHz)
PRR	1	0:9999	Pulse repetition rate (pulses/sec)
BEAM	1	0:3600	Beamwidth (tenths of degrees)
SSECT	1	0:360	Scan sector (degrees)
PFA	1	4 | 6 | 8 | 10	Probability of false alarm (%)
FUNCT	1	SURF | AIR | MISS FIRE | APPRO	Function of radar
RANGE	1	0:500	Maximum detection range (nautical miles)
POWER	1	1:2000	Peak power (dB)
MFUNC	- 4	0:10 alphanumeric 0:500 0:20000 0:9999 0:9999 0:4320 alphanumeric	Multi-function radar parameters: MODE ENOTE, FCODE XCNTD MFREQ MPW MPRR MSRTE STYPE, POLAR
MNFAL	- 1	0:4320	Mean time between failure (minutes)
MNRPR	- 1	0:4320	Mean time to repair (minutes)

Defined type: ESM

Field	Number	Range of Values	Description
HFREQ	1	1:20000	Receiver high freq cutoff (MHz)
LFREQ	1	1:20000	Receiver low freq cutoff (MHz)
LOSS	1	1:2000	System loss (dB)
BRERR	1	0:12	Std development of bearing error (degrees)
MNFAL	1	0:4320	Mean time between failures (minutes)

Object: Ships and Submarines Filename:XXXXXC.SHP

Field	Number	Range of Values	Description
CLASS	1	alphanumeric	Class of the ship or sub
CAT	1	SURF|NSUB|DSUB	Type of ships (defined type)
TYPE	1	alphanumeric	Type of ship or submarine
VMAX	1	0:50	Maximumspeed(knots)
XSECT	1	0:500	Radarcrosssection(dB)
HDG	1	alphanumeric	Name of heading sensor
SPD	1	alphanumeric	Name of speed sensor
CLRNG	1	0:500	Range at which a platform can be classified (miles)
DTRNG	1	0:500	Range at which a platform can be detected (miles)
TRKS	1	0:250	Number of local tracks
BBN	1	0:500	BB @2000 Hz noise at 05/10/15/20/25/30 knots (dB)
BBN3	- 1	0:500	BB @300 Hz noise at 05/10/15/20/25/30 knots (dB)
NBN	- 6	0:999999 0:500	Signature line frequencies (Hz) NB noise (dB)
HFDF	- 1	alphanumeric	HFDF name
DECM	- 1	YES|NO	DECEPTION (RADAR) available?
BLIP	- 1	YES|NO	BLIP enhancment available?
RDESM	- 8	alphanumeric 0:3000	Name and Height of sensor(s) carried (feet)
JAMMR	- 1	alphanumeric 0:3000	Name and Height of jammer(s) carried (feet)
SONAR	- 6	alphanumeric	Name of sonar(s) carried
COMMS	-10	alphanumeric 0:3000	Name and Height of communications suite(s) carried (feet)
WEAP	-30	alphanumeric AAM|ASM|BOMB CIWS|SAM|SSM|GUN 0:10000	Name Type and Qty of weapons carried
MISS	- 4	alphanumeric 0:10000	Types and Qty of cruise missile carried
NDCLS	- 1	1:5	Nuclear Damage Class (1-cv,2-cg,3-dd,4-a,5-sb)

Defined type:SURF

Field	Number	Range of Values	Description
NAV	1	alphanumeric	Name of navigational sensor
LCHRT	1	1:5	Standard flight deck launch rate (aircraft/mibn)
RECRT	1	1:5	Standard flight deck recovery rate (aircraft/mibn)
WAVE	1	0:3000	Wave height threshold for LCHRT and RECRT (feet)
LACOM	1	0:100	Lethal area ratio of comm equipment (%)
LASEN	1	0:100	Lethal area ratio of sensor systems (%)
LAWPN	1	0:100	Lethal area ratio of weapon systems (%)
LAFLD	1	0:100	Lethal area ratio of ship buoyancy (%)
LASPD	1	0:100	Lethal area ratio of ship max speed (%)
LAAIR	1	0:100	Lethal area ratio of parked aircraft (%)
RBOC	1	1:50	Firing rate per minute (rounds/min
		1:10000	Number of rounds (qty)
BUOY	-10	alphanumeric	Name,
		SONO|COMM	Type, and
		0:10000	Number of buoys carried (qty)
AIR	-24	alphanumeric	Name and
		0:10000	Number of aircraft carried (qty)

Defined type:NSUB

Field	Number	Range of Values	Description
DRATE	1	0:1000	Dive rate (ft/min)
KEEL	1	0:3000	Keel depth (feet)
PSCOP	1	0:3000	Periscope depth (feet)
DR	1	alphanumeric	Name of navaigational sensor
OMEGA	1	alphanumeric	Name of navaigational sensor
NAV	- 1	alphanumeric	Name of navaigational sensor
TORP	- 4	alphanumeric	Name and
		0:10000	Qty of torpedoes
WIRES	- 1	0:10000	Wire info for comm
WBUOY	- 4	alphanumeric	Name and
		0:10000	Qty of Wire Buoys for comm
NBUOY	- 4	alphanumeric	Name and
		0:10000	Qty of Non-Wired Buoy for comm

Defined type:DSUB

Field	Number	Range of Values	Description
DRATE	1	0:1000	Dive rate (ft/min)
KEEL	1	0:3000	Keel depth (feet)
PSCOP	1	0:3000	Periscope depth (feet)
DR	1	alphanumeric	Name of navigational sensor
OMEGA	1	alphanumeric	Name of navigational sensor
NAV	- 1	alphanumeric	Name of navigational sensor
TORP	- 4	alphanumeric	Name and
		0:10000	Qty of torpedoes
WIRES	- 1	0:10000	Wire info for comm
WBUOY	- 4	alphanumeric	Name and
		0:10000	Qty of Wire Buoys for comm
NBUOY	- 4	alphanumeric	Name and
		0:10000	Qty of Non-Wired Buoy for comm
BBD	1	0:500	BB @2000 Hz noise at 05/10/15/20/25/30 knots (dB)
BBD3	- 1	0:500	BB @300 Hz noise at 05/10/15/20/25/30 knots (dB)
NBD	- 6	0:999999	Signature line frequencies (Hz)
		0:500	Diesel NB noise (dB)

Object: Shore Bases Filename: XXXXXC.SHR

Field	Number	Range of Values	Description
NAME	1	alphanumeric	Name of the shor^ base
LAT	1	0:90	Latitude in Degrees and minutes
LONG	1	0:180	Longitude in degrees and minutes
JP	1	0:1000000	Amount of aviation fuel (lbs)
LCHRT	1	1:20	Standard aircraft launch rate (aircraft/min)
RECRT	1	1:20	Standard aircraft recovery rate (aircraft/min)
LACOM	1	0:100	Lethal area ratio for communications (%)
LAAIR	1	0:100	Lethal area ratio for aircraft parked (%)
LASEN	1	0:100	Lethal area ratio for sensors on base (%)
LASAM	1	0:100	Lethal area ratio for SAM sites (%)
LASTO	1	0:100	Lethal area ratio for deployable stores (%)
LAJP	1	0:100	Lethal area ratio for aviation fuel (%)
TRKS	1	0:250	Number of local tracks
HFDF	- 1	alphanumeric	Name of HFDF system
RDESM	- 8	alphanumeric 0:3000	Name of sensor Antenna height (feet)
JAMMR	- 1	alphanumeric 0:3000	Name of jammer Antenna height (feet)
BUOY	-10	alphanumeric SONO | COMM 0:3000	Name, Types and Qty of buoys carried
COMMS	-10	alphanumeric 0:3000	Name of commsuite Antenna height (feet)
CJAMR	-4	alphanumeric 0:3000	Name of comm jammer Antenna height (feet)
MISS	- 6	alphanumeric 1:3000	Name and Qty of cruise miss. carried
WEAP	-30	alphanumeric AAM | ASM | BOMB SSM | TORP | CIWS SAM | GUN 1:3000	Name, Types and Qty of weapons carried
AIR	-30	alphanumeric 1:3000	Name and Qty of aircraft

Object: Sonobuoys Filename: XXXXXC.SNB

Field	Number	Range of Values	Description
NAME	1	alphanumeric	Name of Sonar
CAT	1	PASS | ACT | COMM	Type of sonar (defined type)
LIFE	1	0:4320	Buoy life when deployed (minutes)
CFREQ	1	0:20000	Center frequency (Hz)
MNFAL	- 1	0:4320	Mean time between failures (minutes)
MNRPR	-1	1:4320	Mean time to repair (minutes)

Defined type: PASS

Field	Number	Range of Values	Description
LFREQ	1	10:20000	Low end of band frequency (Hz)
HFREQ	1	10:20000	High end of band frequency (Hz)
HZBW	1	0:360	Horizontal beamwidth (degrees)
VRBW	1	0:90	Vertical beamwidth (degrees)
BRERR	1	0:90	Bearing err std dev (degrees)
RD	1	-50:50	Recognition differential (dB)
DI00	1	-30:30	Directivity indices @ 6 speeds 0- 60 degrees (dB)
DI60	1	-30:30	Directivity indices @ 6 speeds 60-120 degrees (dB)
DI120	1	-30:30	Directivity indices @ 6 speeds 120-180 degrees (dB)
OMNI	- 1	YES | NO	OMNI Directional buoy?

Defined type: ACT

Field	Number	Range of Values	Description
LEVEL	1	0:1000	Source level (dB)

Defined type: COMM

Field	Number	Range of Values	Description
LFREQ	1	10:20000	Low end of band frequency (Hz)
HFREQ	1	10:20000	High end of band frequency (Hz)
HZBW	1	0:360	Horizontal beamwidth (degrees)
VRBW	1	0:90	Vertical beamwidth (degrees)
LEVEL	1	0:1000	Source level (dB)
RD	1	-50:50	Recognition differential (dB)
DI00	1	-30:30	Directivity indices @ 6 speeds 0- 60 degrees (dB)
DI60	1	-30:30	Directivity indices @ 6 speeds 60-120 degrees (dB)
DI120	1	-30:30	Directivity indices @ 6 speeds 120-180 degrees (dB)

Object: Active and Passive Sonar Systems Filename: XXXXXC.SNR

Field	Number	Range of Values	Description
NAME	1	alphanumeric	Name of sonar
CAT	1	TOWED | VDS | AHULL PHULL | COMM	Type of sonar (defined type)
CFREQ	1	10:20000	Center frequency (Hz)
MNFAL	- 1	0:4320	Mean time between failures (minutes)
MNRPR	- 1	0:4320	Mean time to repair (minutes)

Defined type: TOWED

Field	Number	Range of Values	Description
LFREQ	1	10:20000	Low end of band frequency (Hz)
HFREQ	1	10:20000	High end of band frequency (Hz)
HZBW	1	0:360	Horizontal beamwidth (degrees)
VERBW	1	0:90	Vertical beamwidth (degrees)
BRERR	1	0:90	Bearing err std dev (degrees)
RD	1	-50:50	Recognition differential (dB)
DI00	1	-30:30	Directivity indices @ 6 speeds 0- 60 degrees (dB)
DI60	1	-30:30	Directivity indices @ 6 speeds 60-120 degrees (dB)
DI120	1	-30:30	Directivity indices @ 6 speeds 120-180 degrees (dB)
CABLE	1	0:99999	Cable Length (yds)
BBN	1	0:300	Broad Band Noise @ 6 speeds (dB)
NBN1	1	0:300	Narrow Band Noise @ 6 speeds freq 10 Hz (dB)
NBN2	1	0:300	Narrow Band Noise @ 6 speeds freq 15 Hz (dB)
NBN3	1	0:300	Narrow Band Noise @ 6 speeds freq 20 Hz (dB)
NBN4	1	0:300	Narrow Band Noise @ 6 speeds freq 30 Hz (dB)
NBN5	1	0:300	Narrow Band Noise @ 6 speeds freq 40 Hz (dB)
NBN6	1	0:300	Narrow Band Noise @ 6 speeds freq 60 Hz (dB)
NBN7	1	0:300	Narrow Band Noise @ 6 speeds freq 80 Hz (dB)
NBN8	1	0:300	Narrow Band Noise @ 6 speeds freq 100 Hz (dB)
NBN9	1	0:300	Narrow Band Noise @ 6 speeds freq 125 Hz (dB)
NBN10	1	0:300	Narrow Band Noise @ 6 speeds freq 250 Hz (dB)
NBN11	1	0:300	Narrow Band Noise @ 6 speeds freq 315 Hz (dB)
NBN12	1	0:300	Narrow Band Noise @ 6 speeds freq 630 Hz (dB)
NBN13	1	0:300	Narrow Band Noise @ 6 speeds freq 900 Hz (dB)
NBN14	1	0:300	Narrow Band Noise @ 6 speeds freq 1250 Hz (dB)
NBN15	1	0:300	Narrow Band Noise @ 6 speeds freq 1600 Hz (dB)
NBN16	1	0:300	Narrow Band Noise @ 6 speeds freq 2000 Hz (dB)
NBN17	1	0:300	Narrow Band Noise @ 6 speeds freq 2500 Hz (dB)
NBN18	1	0:300	Narrow Band Noise @ 6 speeds freq 3500 Hz (dB)
NBN19	1	0:300	Narrow Band Noise @ 6 speeds freq 5000 Hz (dB)
NBN20	1	0:300	Narrow Band Noise @ 6 speeds freq 10000 Hz (dB)

Defined type: VDS

Field	Number	Range of Values	Description
LEVEL	1	0:1000	Source level (dB)

Defined type: PHULL

Field	Number	Range of Values	Description
LFREQ	1	10:20000	Low end of band frequency (Hz)
HFREQ	1	10:20000	High end of band frequency (Hz)
HZBW	1	0:360	Horizontal beamwidth (degrees)
VERBW	1	0:90	Vertical beamwidth (degrees)
BRERR	1	0:90	Bearing err std dev (degrees)
RD	1	-50:50	Recognition differential (dB)
DI00	1	-30:30	Directivity indices @ 6 speeds 0- 60 degrees (dB)
DI60	1	-30:30	Directivity indices @ 6 speeds 60-120 degrees (dB)
DI120	1	-30:30	Directivity indices @ 6 speeds 120-180 degrees (dB)
BBN	1	0:300	Broad Band Noise @ 6 speeds (dB)
NBN1	1	0:300	Narrow Band Noise @ 6 speeds freq 10 Hz (dB)
NBN2	1	0:300	Narrow Band Noise @ 6 speeds freq 15 Hz (dB)
NBN3	1	0:300	Narrow Band Noise @ 6 speeds freq 20 Hz (dB)
NBN4	1	0:300	Narrow Band Noise @ 6 speeds freq 30 Hz (dB)
NBN5	1	0:300	Narrow Band Noise @ 6 speeds freq 40 Hz (dB)
NBN6	1	0:300	Narrow Band Noise @ 6 speeds freq 60 Hz (dB)
NBN7	1	0:300	Narrow Band Noise @ 6 speeds freq 80 Hz (dB)
NBN8	1	0:300	Narrow Band Noise @ 6 speeds freq 100 Hz (dB)
NBN9	1	0:300	Narrow Band Noise @ 6 speeds freq 125 Hz (dB)
NBN10	1	0:300	Narrow Band Noise @ 6 speeds freq 250 Hz (dB)
NBN11	1	0:300	Narrow Band Noise @ 6 speeds freq 315 Hz (dB)
NBN12	1	0:300	Narrow Band Noise @ 6 speeds freq 630 Hz (dB)
NBN13	1	0:300	Narrow Band Noise @ 6 speeds freq 900 Hz (dB)
NBN14	1	0:300	Narrow Band Noise @ 6 speeds freq 1250 Hz (dB)
NBN15	1	0:300	Narrow Band Noise @ 6 speeds freq 1600 Hz (dB)
NBN16	1	0:300	Narrow Band Noise @ 6 speeds freq 2000 Hz (dB)
NBN17	1	0:300	Narrow Band Noise @ 6 speeds freq 2500 Hz (dB)
NBN18	1	0:300	Narrow Band Noise @ 6 speeds freq 3500 Hz (dB)
NBN19	1	0:300	Narrow Band Noise @ 6 speeds freq 5000 Hz (dB)
NBN20	1	0:300	Narrow Band Noise @ 6 speeds freq 10000 Hz (dB)

Defined type: AHULL

Field	Number	Range of Values	Description
LEVEL	1	0:1000	Source level (dB)
MODE	-3	DP | BB | CZ	Allowable operaating mode

Defined type: COMM

Field	Number	Range of Values	Description
LFREQ	1	10:20000	Low end of band frequency (Hz)
HFREQ	1	10:20000	High end of band frequency (Hz)
LEVEL	1	0:1000	Source level (dB)
HZBW	1	0:360	Horizontal beamwidth (degrees)
VRBW	1	0:90	Vertical beamwidth (degrees)
RD	1	-50:50	Recognition Differential (dB)
DI00	1	-30:30	Directivity indices @ 6 speeds 0- 60 degrees (dB)
DI60	1	-30:30	Directivity indices @ 6 speeds 60-120 degrees (dB)
DI120	1	-30:30	Directivity indices @ 6 speeds 120-180 degrees (dB)
BBN	1	0:300	Broad Band Noise @ 6 speeds (dB)
NBN1	1	0:300	Narrow Band Noise @ 6 speeds freq 10 Hz (dB)
NBN2	1	0:300	Narrow Band Noise @ 6 speeds freq 15 Hz (dB)
NBN3	1	0:300	Narrow Band Noise @ 6 speeds freq 20 Hz (dB)
NBN4	1	0:300	Narrow Band Noise @ 6 speeds freq 30 Hz (dB)
NBN5	1	0:300	Narrow Band Noise @ 6 speeds freq 40 Hz (dB)
NBN6	1	0:300	Narrow Band Noise @ 6 speeds freq 60 Hz (dB)
NBN7	1	0:300	Narrow Band Noise @ 6 speeds freq 80 Hz (dB)
NBN8	1	0:300	Narrow Band Noise @ 6 speeds freq 100 Hz (dB)
NBN9	1	0:300	Narrow Band Noise @ 6 speeds freq 125 Hz (dB)
NBN10	1	0:300	Narrow Band Noise @ 6 speeds freq 250 Hz (dB)
NBN11	1	0:300	Narrow Band Noise @ 6 speeds freq 315 Hz (dB)
NBN12	1	0:300	Narrow Band Noise @ 6 speeds freq 630 Hz (dB)
NBN13	1	0:300	Narrow Band Noise @ 6 speeds freq 900 Hz (dB)
NBN14	1	0:300	Narrow Band Noise @ 6 speeds freq 1250 Hz (dB)
NBN15	1	0:300	Narrow Band Noise @ 6 speeds freq 1600 Hz (dB)
NBN16	1	0:300	Narrow Band Noise @ 6 speeds freq 2000 Hz (dB)
NBN17	1	0:300	Narrow Band Noise @ 6 speeds freq 2500 Hz (dB)
NBN18	1	0:300	Narrow Band Noise @ 6 speeds freq 3500 Hz (dB)
NBN19	1	0:300	Narrow Band Noise @ 6 speeds freq 5000 Hz (dB)
NBN20	1	0:300	Narrow Band Noise @ 6 speeds freq 10000 Hz (dB)

Object: Weapons Filename: XXXXXC.WEP

Field	Number	Range of Values	Description
NAME	1	alphanumeric	Name of the weapon
CAT	1	ASM | SAM | AAM CIWS | BOMB | TORP GUN | MINE	Type of weapon (defined type)
MXRNG	1	0:200	Maximum range (miles)
MNFAL	- 1	0:300	Mean time between failure (minutes)
MNRPR	- 1	0:300	Mean time to repair (minutes)
BURST	1	AIR | SURF | SUNSU	Burst type (Nuclear)
DEPTH	1	SHALL | DEEP	Depth of subsurface burst (Nuclear)
YIELD	1	0:32767	Yield of Nuclear burst (kt)

Defined type: ASM

Field	Number	Range of Values	Description
PH	1	0:100	Probability of hit (%)
EPU	1	0:100	Equivalent payload units (lbs)
SALVO	1	1:50	Rounds per salvo
ARM	- 1	YES | NO	Anti radiation missile capability

Defined type: SAM

Field	Number	Range of Values	Description
PK	1	0:100	Probability of Kill (%)
RADAR	1	alphanumeric	Name of radar
SIMUL	1	1:50	Number of simultaneous engagments (qty)
VCRU	- 1	1:2000	Speed (knots)

Defined type: AAM

Field	Number	Range of Values	Description
PK	1	0:100	Probability of Kill (%)
SALVO	1	1:50	Rounds per salvo
FRATE	1	1:50	Firing rate (qty/min)

Defined type: CIWS

Field	Number	Range of Values	Description
PK	1	0:100	Probability of kill (%)
SIMUL	1	1:50	Number of simultanous engagments (qty)

Defined type: BOMB

Field	Number	Range of Values	Description
PH	1	0:100	Probability of hit (%)
EPU	1	0:100	Equivalent payload units (hundreds of lbs)
SALVO	1	1:50	Rounds per salvo

Defined type: TORP

Field	Number	Range of Values	Description
PH	1	0:100	Maximum Prob Hit (%)
RPH10	1	100:60000	Range (yds) for 100% Prob Hit
RPH9	1	100:60000	Range (yds) for 90% Prob Hit
RPH8	1	100:60000	Range (yds) for 80% Prob Hit
RPH7	1	100:60000	Range (yds) for 70% Prob Hit
RPH6	1	100:60000	Range (yds) for 60% Prob Hit
RPH5	1	100:60000	Range (yds) for 50% Prob Hit
RPH4	1	100:60000	Range (yds) for 40% Prob Hit
RPH3	1	100:60000	Range (yds) for 30% Prob Hit
RPH2	1	100:60000	Range (yds) for 20% Prob Hit
RPH1	1	100:60000	Range (yds) for 10% Prob Hit
EPU	1	0:100	Equivalent payload units (hundreds of lbs)
FRATE	1	1:50	Firing rate (rounds/min)
MXALT	1	0:1000	Max altitude for deployment (feet)
VCRU	1	1:2047	Speed (knots)
SUBS	1	YES | NO	Subsurface target only?

Defined type: GUN

Field	Number	Range of Values	Description
RPH10	1	100:60000	Range (yds) for 100% Prob Hit
RPH9	1	100:60000	Range (yds) for 90% Prob Hit
RPH8	1	100:60000	Range (yds) for 80% Prob Hit
RPH7	1	100:60000	Range (yds) for 70% Prob Hit
RPH6	1	100:60000	Range (yds) for 60% Prob Hit
RPH5	1	100:60000	Range (yds) for 50% Prob Hit
RPH4	1	100:60000	Range (yds) for 40% Prob Hit
RPH3	1	100:60000	Range (yds) for 30% Prob Hit
RPH2	1	100:60000	Range (yds) for 20% Prob Hit
RPH1	1	100:60000	Range (yds) for 10% Prob Hit
EPU	1	0:100	Equivalent payload units (hundreds of lbs)
SALVO	1	1:50	Rounds per salvo (qty)
SIMUL	1	1:50	Number of simultanous engagements
VCRU	1	1:2047	Muzzle speed of projectile (knots)

Defined type: MINE

Field	Number	Range of Values	Description
EPU	1	0:100	Equivalent payload units (hundreds of lbs)
MXALT	1	0:1000	Max altitude for deployment (feet)

Appendix B.

BUILD COMMAND GRAMMAR

```
::=      -   "is denoted by"
{ }      -   "zero or more occurances of"
[ ]      -   optional  parameter
( )      -   system provided user prompt
 |       -   "or"
< >      -   -non-terminal symbol
CAPS     -   terminal symbol
...      -   multiple parameters
```

```
<BUILD_order>
  ::=  <program_control_order>
     | <characteristics_control_order>
     | <object_access_order>

<program_control_order>
  ::=  BUILD
     | PRINT [ <characteristics_parameter> ]
     | WRITE [ <characteristics_parameter> ]
     | BYE

<characteristics_control_order>
  ::=  FIND <object_name>
     | SAVE
     | LIST
     | MORE
```

| NAME

| CLASS

| KILL

| DELETE <delete_parm>

| HELP <help_parm_list>

<object_access_order>

::= AIR | COMMBOUY | COMMPAIR | COMMJAMMER | COMMSUITE

| CRUISE MISSILE | JAMMER | NAVAID | RADAR/ESM |

| SURVSAT | SHIP | SHORE BASE | SONAR | SONOBUOY | WEAPON

<characteristics_parameter>

::= AIR | COMMBOUY | COMMPAIR | COMMJAMMER | COMMSUITE

| CRUISE MISSILE | JAMMER | NAVAID | RADAR/ESM | SURVSAT | SHIP

| SHORE BASE | SONAR | SONOBUOY | WEAPON

<delete_parm>

::= ENTRY

| <name_of_characteristic>

<help_parm_list>

::= <help_parm> <help_parm_list>

| <help_parm>

<help_parm>

::= <program_control_order>

| <object_access_order>

| <characteristics_control_order>

| <name_of_characterisitic>

FORCE COMMAND GRAMMAR

<FORCE_order>
::= <program_control_order>
| <scenario_control_order>

<program_control_order>
::= FORCE
| STOP
| BYE

<scenario_control_order>
::= ENTER <force_scenario_parameter>
| DELETE <delete_parameter>
| PRINT <print_parameter>

<force_scenario_parameter>
::= SHIP <task_no> <ship_name> (class) <class> POSITION <latitude>
<longitude> (course) <course> (speed) <speed> DEPTH <depth>

| SHIP <task_no> <ship_name> (class) <class> STATION (ON)
<guide_name> (bearing) <bearing> (RANGE) <range>

| BASE <task_no> <base_name>

| BOAT (boat name) <boat_name> (class) <class> (at)
<base_name/ship_name>

| AIR (maint_log_side_number_id) <id> (seq) <sequence_number>
(quantity) <quantity> (of) <aircraft_type> (base_name)
<base_name> (time_flown) <time_flown> (up_time) <up_time>

| COMMPATH (named) <name> (node) <number> MU <mu> RHO <rho> PAIR
<name>

| COMMJAMMER DEFINITION (minimum j/s) <j/s> (increment)
<j/s_increment>

| COMMJAMMER VALUE (j/s) <j/s> (mu/rho) <mu_rho>

| MEMBER (platform) <force_name> (to_paths) <commpath_1> ...
<commpath_4>

| CIRCUIT <circuit_number> (as commpath) <commpath_name>

| REPORT (policy_for) <BLUE|ORANGE> (named) <policy_name>
(using_circuit) <nr> (or_circuit_nr) <nr> (violate_emcon)
<YES|NO> (intervals_for_position) <mins> (ship_tracks) <mins>
(air_tracks) <mins> (esm_tracks) <mins>

| EMCON (plan_for) <BLUE|ORANGE> (named) <name>
(allow_surface_search) <Y|N> (air_search) <Y|N> (approach)
<Y|N> (airborne_radar) <Y|N> (active_sonar) <Y|N> (hf_comm)
<Y|N> (hf_low_risk) <Y|N> (vhf_or_uhf) <Y|N>

| ESM (fingerprinted_emitter) <emitter_name_1> (for) <ship_name> ...
<emitter_name_4> (for) <ship_name>

| PLAN (for) <BLUE|ORANGE> (named) <plan_name>
{ <valid_wargame_order> } STOP

| CLASSIFICATION (for_ship) <ship_name> (as) <name> (with_tonals)
<tone1> <tone2> <tone3>

| ORDERS { <valid_wargame_order> } STOP

| SEARCH (for) <BLUE|ORANGE> (named) <plan_name>
{ <valid_wargame_order> } STOP

| SONAR (region) <number> (environment) <number> (point_1)
<latitude> <longitude> (point_2) <latitude> <longitude>
(point_3) <latitude> <longitude> ... (point_6) <latitude>
<longitude>

| SOSUS REGION <number> (mean axis length) <nmi> (sigma length)
<nmi> (boundary depth) <feet> (point 1) <latitude> <longitude>
(point 2) <latitude> <longitude> (point 3) <latitude>
<longitude> ... (point 6) <latitude> <longitude>

| SOSUS SUBMARINE <class> (in_region) <nr> (when) <DEEP|SHALLOW>
(prob detect) <pd1> <pd2> <pd3> <pd4> <pd5> <p6> (prob hold)
<p1> <pd2> <pd3> <pd4> <pd5> <p6>

| HFDF (for base) <force_name> (row) <row_number> (prob detect)
<pd1> <pd2> <pd3> <pd4> <pd5> <pd6> <pd7> <pd8> <pd9> <pd10>

| SURVSAT (named) <satellite_name> (of type) <satellite_type>
(ground station) <ground_station_name> (altitude) <nmi>
(duty cycle utilization) <percent> ORBITAL (orbital period)
<minutes> (inclination degrees) <degrees> (minutes) <minutes>
(crossing time) <time> (at longitude) <longitude>

| SURVSAT (named) <satellite_name> (of type) <satellite_type>
(ground station) <ground_station_name> (altitude) <nmi>
(duty cycle utilization) <percent> STATIONARY (at latitide)
<latitude> (and longitude) <longitude>

```
  | WEATHER (for_region) <region_number> (wave_height) <feet>
      (direction) <degrees> (wind_speed) <knots> (direction)
      <degrees> (cloud_cover) <percent> (ceiling) <feet> (depth)
      <feet> (visibility) <nmi> (category) <CLEAR|FOG|HAZE|RAIN>

  | REPORT (policy for) <BLUE|ORANGE> (named) <policy_name> (circuit)
      <circuit_number1> (or) <circuit_number2> (violate EMCON)
      <YES|NO> (intervals for position) <minutes> (ship tracks)
      <minutes> (air tracks) <minutes>

  | REPORT (policy for) <BLUE|ORANGE> (named) <policy_name> (circuit)
      <circuit_number1> (or) <circuit_number2> (violate EMCON)
      <YES|NO> (intervals for position) <minutes> (ship tracks)
      <minutes> (air tracks) <minutes> FORCE <force_name>

  | REPORT (policy for) <BLUE|ORANGE> (named) <policy_name> (circuit)
      <circuit_number1> (or) <circuit_number2> (violate EMCON)
      <YES|NO> (intervals for position) <minutes> (ship tracks)
      <minutes> (air tracks) <minutes> MISSION <AEW ASW | AIRTANKER
      | CAP | DECOY | EW | JAMMER | RECONN | RELAY | RESCUE | SEARCH
      | STRCAP | STRIKE | STTANKER | SURCAP | SURVEILLANCE>

  | PROBHIT (against target type) <name> (with weapon type) <name>
      (% prob hit) <percent>

<delete_parameter>
 ::=  FORCE <force_task_number>
  | PLAN <BLUE|ORANGE> <plan_name>
  | SEARCH <BLUE|ORANGE> <plan_name>
  | ORDERS <order_number1> ... <order_number8>
  | REPORT <BLUE|ORANGE> <policy_name>
```

| COMMPATH <commpath_name>

| MEMBER <platform_name> <commpath_name1> ... <commpath_name4>

| EMCON <BLUE|ORANGE> <EMCON_plan_name>

| HFDF <base_name>

| SURVSAT <survsat_name>

| SOSUS REGION <region_number>

| SOSUS SUBMARINE <sub_class> <region_number>

| WEATHER <region_number>

| PROBHIT <target_type> <weapon_type>

| AIR <side_number_id> <beginning_seq_number> <quantity>

| ESM <emitter_name1> <platform_name1> ... <emitter_name4>
<platform_name4>

| COMMJAMMMER DEFINTION

| COMMJAMMER VALUE <js>

<print_parameter>

::= ALL FORCES | UTILIZATION | COMMPATH | ESM | ORDERS | PLAN |
SEARCH

| REPORT | AIR | EMCON | HFDF | SOSUS | SURVSAT | WEATHER

| COMMJAMMER

APPENDIX C

BATMAN AND ROBIN SCENARIO STRUCTURE

```
#define NUM_THEATERS 12
#define NUM_SCEN_PER_THEATER 25
#define NUM_PLATS_PER_FORCE 150
#define MAX_NUM_STUDENTS 100

typedef struct plat_pair {
        char        name[30];
        int         number;
} PLAT_PAIR;


typedef struct theater_files {
        char        scenario_location_name[80];
        int         num_scenarios;
        int         scenario_num[NUM_SCEN_PER_THEATER];
} THEATER;


typedef struct scenario_head {
        struct theater_files        all_theaters[NUM_THEATERS];
        struct plat_pair            red_plats[NUM_PLATS_PER_FORCE];
        struct plat_pair            blue_plats[NUM_PLATS_PER_FORCE];
        int         scenarios_avail[NUM_THEATERS * NUM_SCEN_PER_THEATER];
        char        *student_names[MAX_NUM_STUDENTS];

        int         game_mode;                  /* BATMAN or ROBIN */
        int         engine_state;               /* RUNNING, PAUSED or STOPPED */
        int         red_plats_active;           /* TRUE or FALSE */
        float       engine_update_interval;
        int         engine_loop_count;
        int         canvas_down_click_x;        /* the x coord of the last click is saved */
                                                /* here because the canvas_ie positions */
                                                /* change too fast for some applications */
        int         canvas_down_click_y;        /* the y coord of the last click is saved */
                                                /* here because the canvas_ie positions */
                                                /* change too fast for some applications */
        int         canvas_down_click_code; /* the event ie_code is saved */
                                                /* here because the canvas_ie positions */
                                                /* change too fast for some applications */

        int         time_to_stop;
        LIST_HEAD                   timer_nodes;
        LIST_HEAD                   playback_nodes;

        int         ext_game_phase;             /* LOADOUT, MAIN, SETUP, GRID, or STATS */
        char        master_password[STRING_LENGTH];
        char        player_name[STRING_LENGTH];
        char        theater_rf[STRING_LENGTH];
        char        theater_name[STRING_LENGTH];
        int         theater_index;
        int         sp_vlx,sp_vly;
        int         ximage, yimage;             /* The x,y position for the pixwin */
        int         map_range;                  /* The current zoom factor of the map */
```

```
int          grid_radius;              /* The current grid radius for the map */
double       vl_lat,vl_lon;
int          randomize;                /* Random scenario selection flag */
int          scen_num;                 /* The current scenario number */

int          still_in_last_engine_call;

struct engine_node          *input_nodes;
struct engine_node          *curr_node;
struct engine_node          *map_copy;
struct engine_node          *clock;
struct engine_node          *update_plat;
struct engine_node          *red_detect_build;
struct engine_node          *red_detect_plat;
struct engine_node          *blue_detect_build;
struct engine_node          *blue_detect_plat;
struct engine_node          *draw_plat;
struct engine_node          *update_misc;
struct engine_node          *display_changes;
Frame        mainframe;
Rect         *icon_rect;
Rect         *frame_rect;
Canvas       maincanvas;
Rect         *maincanvas_rect;
Pixwin       *canvas_pw;
int          canvas_fd;
Event        *canvas_ie;
short        status_on;
short        main_status_on;
CANVAS_WIN                  *status_canvas;
ENGINE_NODE                 *main_status_display_node;
struct pr_prpos             status_title;
PANEL_WIN                   *curr_pan_win;

/* robin usage */
PANEL_WIN                   *editor_panel_win;
PANEL_WIN                   *manager_panel_win;
PANEL_WIN                   *assignments_panel_win;
PANEL_WIN                   *class_panel_win;
PANEL_WIN                   *test_panel_win;
PANEL_WIN                   *defcon_panel_win;

CANVAS_WIN                  *map_canvas_win;

int          depth;
int          disp_grid_in_batman;
int          debug_on;
int          playback_state;
int          stats_save_events;
int          stats_save_results;
int          heads_up_dist;
float        heads_up_dist_squared;

Pixrect      *screen_pr;
Pixrect      *display_pr;
Pixrect      *star_field_pr;
Pixrect      *map_pr;
Pixrect      *robin_view_pr;
Pixrect      *confidence_pr;
u_int        chain_pw_draw_op;
u_int        chain_pw_erase_op;
```

```
	u_int		chain_pr_draw_op;
	u_int		chain_pr_erase_op;
	u_int		map_zoom_op;
	u_int		warning_panel_op;

	Pixfont		*med_font, *sm_font, *big_font, *scale_font;
	short		Integer_input;
	short		Input_entered;
	short		zoomed;
	short		Air_radar_on;
	short		Surf_radar_on;
	short		Subsurf_radar_on;
	struct console_force_head		*cfh;
	struct path_force_head			*pfh;

	Pixrect		*vl_pr;
	Pixrect		*air_explo_pr;
	Pixrect		*surf_explo_pr;
	Pixrect		*solid_pixrect;
	Pixrect		*checks_pixrect;
	Pixrect		*grey25_pixrect;
	Pixrect		*grey75_pixrect;
	Pixrect		*chaff_pixrect;
	Pixrect		*hammer_pixrect;
	Pixrect		*status_pixrect;
	Pixrect		*alert_pixrect;
	Pixrect		*loadout_map_pixrect;
	Pixrect		*lat_lon_pixrect;
	Pixrect		*grid_pr;
	Pixrect		*launch_return_pr;
	Pixrect		*air_pixrect;
	Pixrect		*surf_pixrect;
	Pixrect		*subsurf_pixrect;
	Pixrect		*air_radar_pixrect;
	Pixrect		*surf_radar_pixrect;
	Pixrect		*sonar_pixrect;
	Pixrect		*zoom_pixrect;
	Pixrect		*move_pixrect;
	Pixrect		*remove_pixrect;
	Pixrect		*clear_pixrect;
	Pixrect		*dummy_pixrect;
	Pixrect		*next_types_pr;
	Pixrect		*task_force_pr;
	Pixrect		*cap_pixrect;
	Pixrect		*chain_pixrect;
	Pixrect		*air_status_pr;
	Pixrect		*surf_status_pr;
	Pixrect		*sub_status_pr;
	Pixrect		*tf_status_prs[3];

} SCENARIO_HEAD;
```

LIST OF REFERENCES

Baecker, R.M., and Buxton, W.A.S, *Readings in Human-Computer Interaction: A Mulitdisciplanary Approach*, Morgan Kaufman Publishers, Inc., 1987.

Federico, PA., *BATMAN(Battle-Management Assessment System) & ROBIN(Raid Originator Bogie Ingress): Rationale, Software Design, and Database Descriptions*, Navy Personnel Research and Development Center, 1989.

Naval Ocean Systems Center (NOSC), *BUILD Program User's Guide*, Version 5.0, 1988.

Naval Ocean Systems Center (NOSC), *FORCE Program User's Guide*, Version 5.1, 1989.

Pressman, R.S., *Software Engineering: A Practioner's Approach*, McGraw-Hill Book Company, 1987.

Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley, Publishing Co., 1987.

Shu, N., *Visual Programming*, Van Nostrand Reinhold Company, Inc., 1988.

Stroustrup, B., "What is Object-Oriented Programming?," *IEEE Software*, pp. 10-20, May 1988.

Sun Microsystems, Inc., *SunView Programmers Guide*, 1989.

Sun Microsystems, Inc., *Sunview System Programmers Guide*, 1989.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Commandant of the Marine Corps Code TE 06 Headquarters, U.S. Marine Corps Washington, D.C. 20380-0001	1
4.	Dr. James Eagle Code OR/ER Naval Postgraduate School Monterey, CA 93943	2
5.	CAPT Paul Bloch Code ORBL Naval Postgraduate School Monterey, CA 93943	5
6.	Capt Thomas Avey MCOTEA MCCDC Quantico, VA 22134	2
7.	OP 73 Navy Department The Pentagon Washington, D.C. 92152-5000	1
8.	John Dickinson Code 454 Naval Ocean Systems Center San Diego, CA 92152-5000	3